DISSERTATION

Convergence of Real-time Audio and Video Streaming Technologies

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

o. Univ. Prof. Dipl.-Ing. Dr. techn. Dietmar Dietrich,
Institut für Computertechnik
Technische Universität Wien

und

Prof. Dr. Gerhard P. Hancke
Department of Electrical, Electronic and Computer Engineering
University of Pretoria

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von

Dipl.-Ing. Manfred Weihs
Matrikel-Nr. 9525662
Babogasse 3, 2020 Hollabrunn

Wien, Oktober 2006 _____

Kurzfassung

Die digitale Vernetzung von elektronischen Geräten im Heimbereich gewinnt immer mehr an Bedeutung. In einem typischen Haus werden viele verschiedene Arten von Netzwerken installiert, die für unterschiedliche Zwecke optimiert sind. Einige davon können zur Echtzeit-Übertragung von Audio-/Video-Daten verwendet werden. Gleiches gilt für den Automobil-Bereich. Diese Arbeit beschreibt Konzepte zur Echtzeit-Übertragung von Multimedia-Daten im Heim- und Automobilbereich, geht auf die notwendigen Voraussetzungen ein und erläutert Möglichkeiten, die Grenzen zwischen verschiedenen Netzwerken zu überwinden.

Aufgrund der Anzahl an verschiedenen Netzwerken in diesen Umgebungen, die auch für Audio-/Video-Streaming eingesetzt werden, kann eine Verbindung dieser Netzwerke den Nutzen der angeschlossenen Geräte erhöhen. Es werden Möglichkeiten aufgezeigt, verschiedene Netze zu koppeln, um Quellen und Senken dieser Ströme in unterschiedlichen Arten von Netzwerken zu verbinden. Das Design eines Gateways wird erarbeitet, wobei besonderer Wert darauf gelegt wird, dass dieses möglichst universell ist und an verschiedene Netzwerke und Protokolle angepasst werden kann. Das Gateway führt so geringe Modifikationen wie möglich an dem Multimedia-Datenstrom durch, sodass die Qualität der Wiedergabe möglichst wenig unter der Umsetzung zwischen den verschiedenen Netzwerken leidet. Die Datenübertragung erfolgt auf allen beteiligten Netzwerken gemäß offenen Standards, um einer maximalen Anzahl von Geräten die Teilnahme daran zu ermöglichen. Proprietäre Protokolle werden vermieden.

Im Rahmen von Beispiel-Implementierungen werden Teilaspekte der Problemstellung gelöst und somit die Funktionalität des gewählten Ansatzes nachgewiesen. Spezielle Probleme, die berücksichtigt werden müssen, wie insbesondere das korrekte Zeitverhalten, werden aufgezeigt und Lösungsmöglichkeiten vorgestellt. Es wird gezeigt, wie verschiedene Netzwerke über ein universelles Gateway für Echtzeit-Multimedia-Datenströme gekoppelt werden können.

Abstract

Networking between electronic devices within the home becomes more and more common. In a typical house, there are many different kinds of network that are optimised for dedicated applications. Some can be used for real-time transmission of audio/video data. The same applies to the automotive sector. This work describes concepts for real-time transmission of multimedia data within the home and vehicles, specifies the requirements and outlines how to overcome the borders between the networks with respect to audio/video streaming.

Due to the number of different networks that are used for audio and video streaming in these environments, interconnection between these networks can add benefit to the attached devices. Possibilities for coupling of different networks are examined to connect sources and sinks of these streams in various kinds of network. The design of a gateway is developed. Special attention is paid to make it as universal as possible and allow adaptation for different networks and protocols. The gateway modifies the multimedia data stream as little as possible in order to minimise the degradation of the presentation quality. Data transmission is performed according to open standards on all networks in order to enable as many devices as possible to take part. Proprietary protocols are avoided.

Example implementations demonstrate the solutions to different parts of the problem and prove the functionality of the chosen concept. Special problems that have to be carefully considered, particularly timing issues, are explained, and solutions are proposed. It is shown how different networks can be coupled via a universal gateway for real-time multimedia streaming.

Preface

Multimedia streaming and the IEEE 1394 bus (FireWire) have been one of my major fields of interest already during my diploma studies. My diploma thesis was about reception of digital TV programs as MPEG-2 transport streams over IEEE 1394, decoding them in software and presenting them on an ordinary PC. So I have been in touch with all the protocols involved in real-time transmission of audio and video on IEEE 1394 since the year 2000. After finishing my diploma studies, I became member of the Institute of Computer Technology (ICT) of Vienna University of Technology. During my first year at the ICT, I was involved in the EU funded project InHoMNet, in which a home network concept based on IEEE 1394 was developed. The task of the team at the ICT was to develop an intelligent external tuner receiving digital TV via satellite and transmitting it on IEEE 1394. In particular, the hardware of the system and the software for a HAVi protocol stack had to be developed.

At this time I already realised that real-time streaming of multimedia in home networks is a very interesting research topic. IEEE 1394 would not be the only network in the home that comprises sources and sinks of multimedia data. So it became obvious that coupling of different networks could add a significant benefit for the users. As it turned out, relaying a real-time multimedia data stream from one network into another one causes a bunch of unsolved problem, especially with regard to correct timing. This was my motivation to study this topic in-depth and start writing my dissertation about it.

After the InHoMNet project, I founded the spin-off company "dmn Software-Entwicklung GmbH" together with four colleagues, which focused in developing and marketing of a HAVi protocol stack used for control of audio and video devices on an IEEE 1394 bus. I also got involved into a second EU funded project called HOME-PLANET. We had to provide the HAVi stack and a graphical user interface. In addition, we had to manage streaming connections on IEEE 1394 and decode received audio/video streams. So I stayed in touch with multimedia streaming and, in particular, the protocols used on IEEE 1394 throughout my employment at the ICT and could use many results for this dissertation.

I want to thank the head of ICT, Prof. Dietmar Dietrich, not only for giving me the opportunity to work and perform my doctoral studies at this institute, but also for support and advice concerning this thesis. I thank him and Prof. Gerhard P. Hancke from University of Pretoria for reviewing this dissertation.

I have to express my gratitude to the members of the former "IEEE 1394 multimedia group" at the ICT. Together with Peter Skiczuk, who shared my room for several years and gave me an excellent working atmosphere, Michael Ziehensack, Christian Gillesberger, and René Bauer, I had the pleasure to be part of a very good team. We had many interesting discussions, did efficient and good development, and we founded the company "dmn Software-Entwicklung GmbH".

I also have to thank the members of the teams at the ICT I joined after the IEEE 1394 multimedia group had been closed, specifically Peter Palensky, Brigitte Lorenz, Maxim Lobashov, and Friederich Kupzog. All of them gave me valuable suggestions for this dissertation, regarding content, formal aspects, or procedure, and formed teams I enjoyed to be part of.

Special thanks go to Patrick Loschmidt. We already did most of our diploma studies together, and also during the doctoral studies, he always gave me advice when it was needed. I am indebted to him for proof-reading major parts of this work.

I am grateful to all my students who did their diploma theses or practical courses under my supervision. Some results were valuable input for the work on this dissertation.

Last, but not least, I express my sincere thanks to my family, in particular to my parents, for supporting me and providing me an environment in which academic studies can be carried out.

Contents

1 Introduction

Digital networks have become more and more common in home environments during the last decade. Now they are an integral part of modern home wiring. While in previous times there have been dedicated wires for telephone, TV distribution, computer networking and power supply, these borders blur a bit. There are technologies to distribute digital data for computer networking over power line or television cables, to transmit voice over computer networks, to send TV programs via computer networks, and to use telephone lines to interconnect computers. Customers want to reuse existing infrastructure for additional services and try to avoid additional costs and effort for extra wires. Hence, convergence between different kinds of network, analogue and digital ones, data and power distribution networks, is a very interesting, important, and current topic. There is a desire to use one single cable for as many services as possible and avoid parallel cabling. The network bandwidth available in typical home environments has been increasing steadily over the last couple of years. Therefore, it is feasible to distribute even high bandwidth multimedia content like video and audio digitally within the home using these networks.

It turns out that the same is true for the automotive sector. In addition to the fieldbus systems used to connect sensors and actuators, modern high-end cars have integrated networks that are used for the entertainment system, in particular for distribution of multimedia data. The requirements and also the used technologies are quite similar to the ones used in home environments. Therefore, this thesis addresses real-time multimedia streaming in both, home and automotive environments.

One example for a digital home network that is used as a backbone for various services has been given in the EU funded project InHoMNet [1]. In this project, an IEEE 1394 bus is used as a backbone, and multimedia devices, like a digital satellite tuner and a multimedia TV, a home automation system based on a domotic controller, as well as office devices like PCs and peripherals are integrated. This system architecture is shown in Figure 1.1. The Institute of Computer Technology of Vienna University of Technology had—besides the role of the project coordinator—the major task in multimedia streaming. It developed the prototype of the digital intelligent external tuner, which can receive digital television from satellite and distribute it on the IEEE 1394 bus. More details about this system are given in Section 6.1. The author of the present thesis had his focus on the reception of the digital TV data on the IEEE 1394 bus—which was the subject of his diploma thesis—and supported other project partners in this area. In fact, this work in the InHoMNet project was the starting point for the present dissertation. There was a second EU funded project the author of this thesis was directly involved, the project HOME-PLANET [2]. This project demonstrated an in-home digital network based on IEEE 1394b. The main objective of this project was the development of a home network based on plastic optical fibre (POF). The task of the Institute of Computer Technology—and the author of the present thesis—was the implementation of the necessary software, a control stack and a graphical user interface based on the HAVi protocol with integration of a reception module to present MPEG-2 transport streams received over IEEE 1394. More information about the integration of this MPEG-2 transport stream reception module is given in Section 6.2.

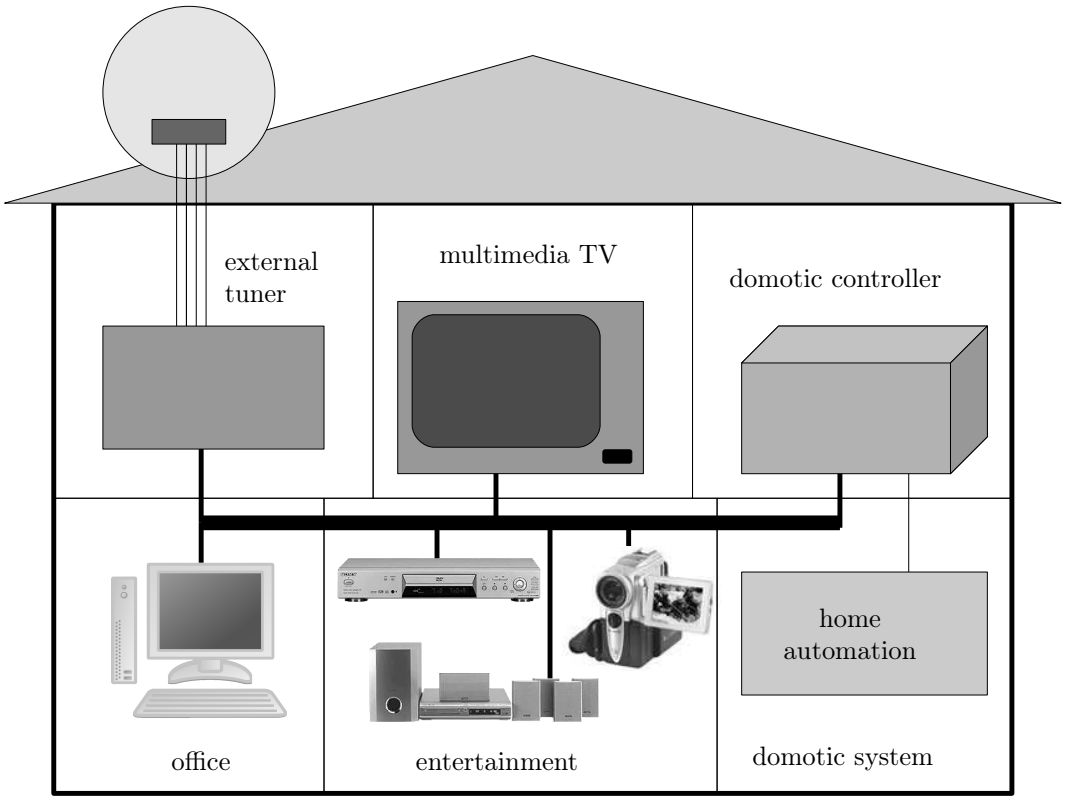

Figure 1.1: System architecture of the InHoMNet project

Although there are different definitions of the term multimedia and generally it is seen as the integration of different kinds of media, including text, animations, still images, etc., in this work the term multimedia is particularly used for audio and video data [3, 4].

1.1 Kinds of Multimedia Streaming

Multimedia data can be transmitted in analogue or in digital form. In this work, streaming refers to digital data transmission of multimedia data in real-time. This has several advantages over analogue transmission. Besides the fact that digital data transmission is not susceptible to quality degradation that always happens in analogue data transmission, the main advantage is that different kinds of data can be integrated within one network. As pointed out above, this is also possible in some cases on analogue networks to some degree. However, digital networks offer much more flexibility in this regard. Furthermore, due to the use of data compression methods, digital transmission allows the transport of much more media content over the same cable than analogue data transmission does. When multimedia data is to be transported over digital networks, the analogue data has first to be converted into digital data, a sequence of bits and bytes. This multimedia data can be distributed in several ways over digital networks. One is "download and play". That means, a file containing multimedia data is completely transferred to the target, which in turn starts playing the file after it has been received completely. The advantage of this mechanism is that it is very easy to implement, common protocols like FTP or HTTP (on TCP/IP networks) can be employed to transfer the file, and there are hardly any

constraints put on the network. Neither throughput nor transit delay have to fulfil requirements. Furthermore, there is no need to adapt the data rate and consequently the quality in order to accommodate variations in the available network bandwidth. The content can be encoded using a fixed or variable data rate according to the quality requirements. The main disadvantage is that the user has to wait until the complete file has been transferred, which can be quite long depending on the size of the file and the available bandwidth of the network. If this latency is seen as one quality parameter of the service, the perceived quality of service by the user might be very bad. Another disadvantage is that the target device has to store the complete file and, consequently, needs considerable storage space, which is a problem for simple, low-cost devices. Therefore, this technique can only be used for content of limited size like short video clips. It is obviously not suited for live content, which is produced in parallel to transmission.

A variation of this approach is "progressive download", sometimes referred to also as "pseudo streaming" or "fast start". It is similar to "download and play", but playback starts before the complete file has been transferred. This only works well if the transmission is at least as fast as playback. Then the necessary delay for start of playback after the start of transmission is constant and might be short, depending on the parameters of the transmission channel, in particular the interarrival jitter. Otherwise, the required delay is proportional to the length of the multimedia stream and also grows linearly with the reciprocal of the network bit rate[1]. If playback starts too early, there will be disruptions. A requirement for this streaming technique is that parts which are to be played together are located together within the file. For example, a file format that stores video at the beginning and audio at the end of the file is not usable for this kind of streaming. The advantages of this streaming method compared to "download and play" are that the initial delay is smaller and the sink device can throw away data after it has been presented and, therefore, requires less storage space. The problems are possible disruptions in case of too early start of playback or network failures as well as long delay and storage requirements in case of network data rates lower than the media data rates.

The third way of distribution of multimedia data is "real-time streaming". If just "streaming" is mentioned, it usually refers to this kind of streaming. This means that multimedia data is transmitted at the same (at least average) speed as it should be presented at the sink device, the stream is transmitted in "real-time". In contrast to "download and play" the target device does not have to store a complete multimedia file, but it only buffers small parts in advance to presentation and discards them afterwards. Therefore, it is usually not possible to replay a scene or to pause. The content can either be live (radio, television) or on demand (e. g. video clips or movies). Video on demand could be combined with pay-per-view. On networks not providing Quality of Service (QoS) it might be necessary to adapt the data rate yielding variations in the quality of the presentation. The main advantage is the inexistent requirement for large storage space. The disadvantages are the requirements the network has to fulfil.

In this work only "real-time streaming" is addressed. The other kinds of streaming are not of interest here because they involve just the usual data transfer protocols and normal data transfer between different networks is a well-known area not requiring much further research. Moreover, many consumer electronic devices do not offer the storage space necessary for these other kinds of streaming. There should be no need to buffer big parts of the multimedia content in the sink devices of the stream. The content distributed over the network is either live content that has to be transmitted in real-time, or it is provided by some server device on demand that

[1] This is true for multimedia data encoded using constant bit rate. For variable bit rate streams the relevant bit rate is between the average bit rate and the maximum bit rate.

transmits it in real-time on the network without additional delay caused by intermediate storing of the content.

1.2 Problem Description

There are many different kinds of network in home and automotive environments, which are used for different purposes and are more or less suitable for streaming of multimedia data. One kind of network often installed in modern homes or vehicles are field bus systems, for instance EIB, LonWorks, CAN, etc. [5, 6]. These are used for connections of sensors and actuators, for example in home automation. They accomplish control and monitoring purposes, offer rather low bandwidth and are, therefore, not well suited for multimedia streaming. Nevertheless, very limited audio streaming can be performed [7].

Another kind of network found in home environments is IEEE 1394, also known as FireWire, which is integrated in many consumer electronic devices like cameras, but also personal computers and peripherals [8]. This network technology is perfectly suited for multimedia streaming and will be evaluated more in detail later.

The most widespread class of networks installed in homes are IP based networks, mostly based on Ethernet [9]. They are usually used to connect personal computers, but also some consumer electronic devices can be integrated into these networks. Most networks are still based on version 4 of the Internet protocol (IP), but the successor (version 6) offerers features that make it more suitable for streaming.

Wireless technologies like Bluetooth or wireless LAN according to the series of IEEE 802.11 have become very popular because they do not require installation of cables. Some wireless technologies provide too little bandwidth to qualify for multimedia streaming purposes. A general problem of wireless networks is that the available resources can vary very much over time due to interference, fading, motion of devices etc. Nevertheless, as it will be shown later, there are wireless solutions providing sufficient bandwidth for real-time multimedia streaming.

A network used in the automotive sector is MOST. It is perfectly suited for real-time streaming of media. In fact, it was designed for this purpose. It offers similar features like IEEE 1394, but its typical application area is limited to the automotive sector, although it can be used in other environments as well [10, 11]. Other kinds of network will be mentioned later.

On all these networks different protocols are used, and different data formats for multimedia are common. All these various networks are usually operated in parallel, and each is used for a dedicated set of purposes. Interconnecting these networks can yield a great benefit for the user. This is especially true for real-time multimedia streaming. Different networks can be used for multimedia streaming and are currently used for multimedia streaming. But these streams are only available within one particular network. Figure 1.2 illustrates this situation: Two uncoupled networks contain sources and sinks for multimedia streams, but streaming connections are only possible within these networks. Only a few connections are possible, some devices might even be unable to participate in streaming connections because appropriate sinks or sources are only available in the other network. Consequently, each device that should present a multimedia stream has to be connected to the same network as the source. For example, if TV content is distributed inside the home via coaxial TV cables and a user wants to watch it on his personal computer, this computer has to be connected to the TV system using an appropriate tuner hardware. If TV is distributed via IEEE 1394, the PC will have to be connected to the IEEE 1394 bus. However, the PC might already be connected to Ethernet, and the user might

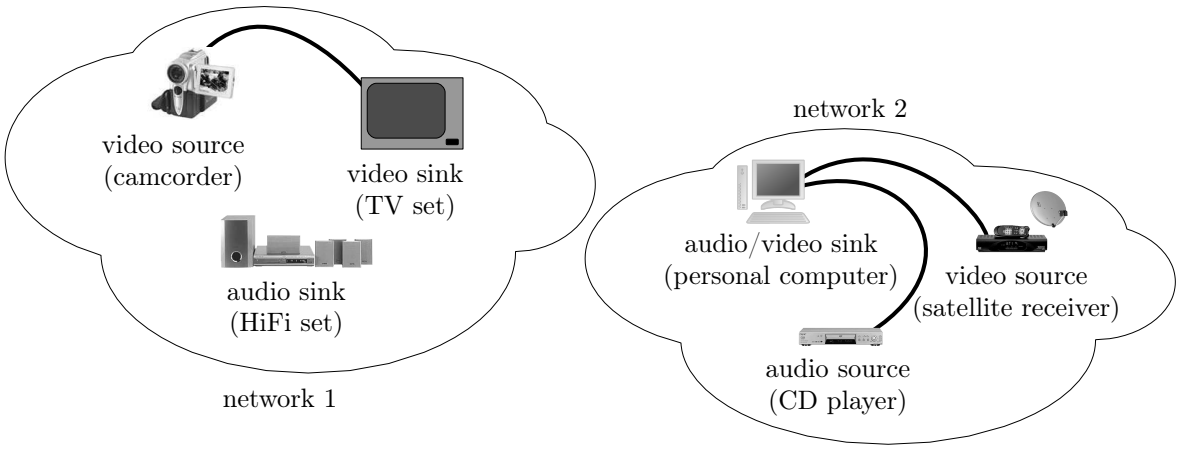

Figure 1.2: Two uncoupled networks containing sources and sinks for multimedia: Only a few streaming connections are possible, some devices cannot participate at all.

want to avoid connecting it to an additional cable. It might even be impossible to install this additional cable for some, e. g. constructional reasons. Therefore, it should be possible to couple the TV distribution network, on whatever technology it is based, with the existing Ethernet and transmit the TV programs also over this network. Convergence between different networks for real-time multimedia streaming can help reducing the wiring required to achieve desired services.

On the other hand, for a given network environment, additional services can be made available. For example, an IEEE 1394 enabled audio system might be able to play media available locally, like CDs, audio cassettes, or radio programs received with an internal tuner, and media available on the IEEE 1394 bus. Consumer electronic devices are usually not extendable like personal computers, so it is not even possible to extend them with an additional network interface and connect them to additional networks. However, there are streams available on other networks, for example live audio streams of some radio stations on the Internet. The user might want to listen to these streams on the audio system in the living room instead of the PC in the office. For this purpose, it is necessary to couple the IEEE 1394 bus with the computer network and make the streams from the Internet also available on the IEEE 1394 bus.

Since there are many different networks in a home and also in an automobile that provide sources and sinks of multimedia streams, a very interesting topic is coupling of various networks. It would be a great benefit for a user if a sink of multimedia, like for instance an audio amplifier, cannot only present streams from sources within the same network, but also streams coming from sources in other networks. Figure 1.3 shows that coupling two networks makes many additional streaming connections possible.

Solutions for streaming of multimedia data over various networks exist. The main problem is the fact that most of them focus on one special application and are limited to one single kind of network. Solutions coupling different networks often use an intermediate network just as transit network or use proprietary protocols not allowing other devices within this network to use the multimedia data. Therefore, the main focus of this work is to develop a universal architecture for a multimedia streaming gateway and demonstrate the possibility to couple networks in a way that as many devices as possible can take advantage of the offered services.

For this purpose of coupling different networks for real-time multimedia streaming, gateways

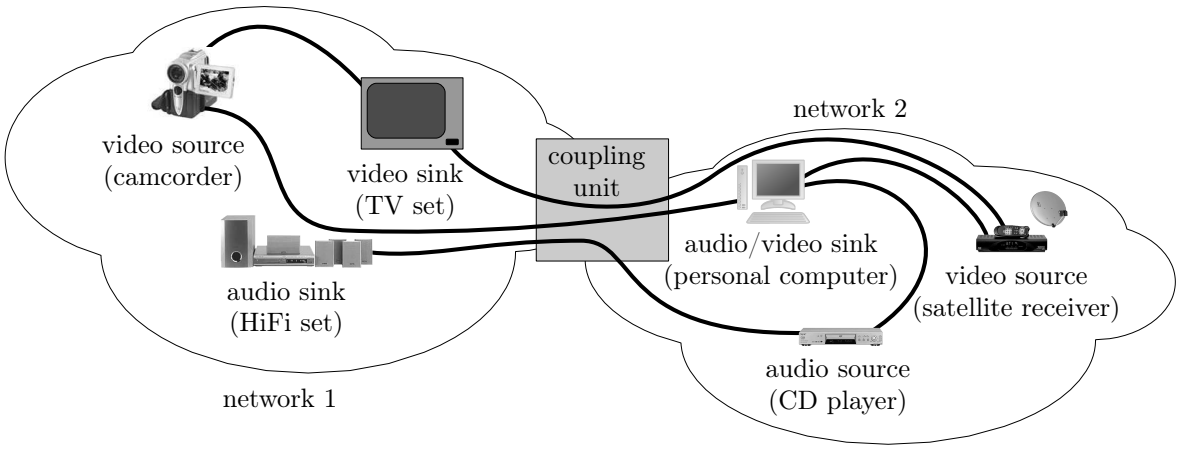

Figure 1.3: Two coupled networks containing sources and sinks for multimedia: Many additional streaming connections are made possible by the coupling unit.

are necessary. These gateways should be universal in order to make as many services as possible from one network available to devices of the other network. One difficulty is that very diverse and incompatible protocols are used on various networks, for control purposes as well as for transport of multimedia data. Also different data formats are used. Because the networks offer different features, it is often not possible to forward multimedia data unmodified. It has be transformed into a format that can be used on the target network.

The aim of this work is to couple different networks and make multimedia content available within one network also available to devices in another network. Sources and sinks of real-time multimedia data should be connectable arbitrarily. For this purpose the data must be transmitted on each network in a form that is understood by the devices in this network. The standards for data transmission that are common on each network should be fulfilled. On the other hand, the gateway should modify the stream as little as possible. It should try to avoid unnecessary degradations in quality. It should not change the timing of the stream and forward it with as little delay as possible.

The biggest difficulty is the heterogeneity of the environment the gateway has to handle. There are so many different protocols and data formats in use, and the networks differ very much in their properties. The gateway should be able to handle streams for each device in a form that it understands. Consequently, such a gateway should be extensible and allow easy integration of support for additional protocols and data formats.

2 Network Requirements for Real-time Multimedia Streaming

A network that should be used for multimedia streaming has to fulfil several requirements. Some are absolutely essential, whereas others can be a bit relaxed. In this chapter the term "Quality of Service" is discussed and the requirements of a network for real-time multimedia streaming are investigated.

2.1 Quality of Service

There are many different interpretations of the term "Quality of Service" (QoS), some are given here, more can be found in [12]. These are some definitions of the term QoS found in literature:

- "Quality of service: A set of quality requirements on the collective behaviour of one or more objects. Quality of service may be specified in a contract or measured and reported after the event. The quality of service may be parameterized. NOTE – Quality of service is concerned with such characteristics as the rate of information transfer, the latency, the probability of a communication being disrupted, the probability of system failure, the probability of storage failure, etc." [13]
- "QoS is the collective effect of service performance which determines the degree of satisfaction of a user of the service." [14]
- "QoS is a set of user-perceivable attributes which describe a service the way it is perceived. It is expressed in a user-understandable language and manifests itself as a number of parameters, all of which have either subjective or objective values." [15]
- "Quality of service represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of an application." [16]

The exact formulations of these definitions differ slightly, some are a bit imprecise. For example, only the third definition requires that QoS must be expressed in a user-understandable language, while the other definitions do not mandate this. All definitions except the second one state that QoS can be described by a set of parameters; according to the fourth definition, they can be qualitative or quantitative, while the examples given in the first definition are all quantitative parameters. This leads to the adverse fact that also the understanding of the term QoS in the scientific community varies slightly. The most authoritative definition is the first one as it is taken from an international standard. The general meaning—which is used in the present work—is that the quality of services, which can relate to services on different layers of a network protocol stack, is characterised by some parameters. Which parameters are used depends on the type of network and service. A discussion of QoS parameters can be found in [16].

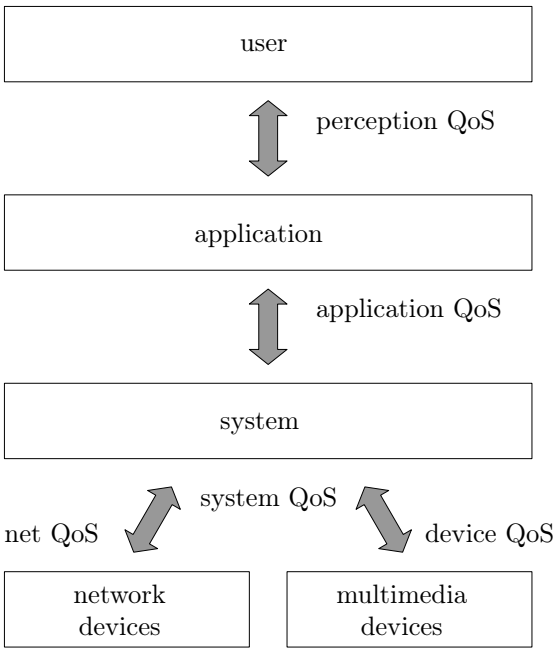

Figure 2.1: Layer model for QoS according to [3]

An in-depth discussion of the term QoS is given in [3]. The author points out that QoS has to be considered on different layers and, therefore, introduces four layers with their associated QoS (see Figure 2.1). This layer model is similar to the OSI (open systems interconnection) reference model [17, 18]. But the OSI reference model is a model for for communication systems, and this is a different model for multimedia systems, which not necessarily involve communication:

- User—perception QoS
- Application—application QoS
- System (including communication services and operating system services)—system QoS
- Devices (network and multimedia devices)—device QoS, net QoS

This is one way to classify different kinds of QoS. Of course, there are also others. The QoS perceived by the user is usually characterised by qualitative parameters like audio quality (telephone quality, CD quality, etc.), optical quality of images, font readability, but also quantitative parameters like skew between audio and video in an A/V presentation or delay in case of interactive media. On the lower layers the parameters are more quantitative (like throughput, packet size, etc.). QoS parameters can be specified in a deterministic way (e. g. maximum delay) or a statistic way (e. g. bit error rate).

There exists a mapping between QoS parameters on the various layers. For example, the requirement to perceive video at VHS quality (QoS on user layer) puts certain constraints on QoS parameters on application layer (resolution, frame rate, some coding parameters). On the other hand, they can be translated into some system QoS parameters like data rate, latency, or CPU power. And finally, they will result in some network QoS (packet size, bit error rate, delay, etc.).

The translation between QoS parameters on different layers is not always unique. To achieve a certain throughput, it is possible to tune packet size and/or packet rate. And in the other direction, if the throughput is too little, either resolution or frame rate can be reduced. Therefore, these translations require some knowledge or strategy on how to map QoS parameters of one layer to those on another one.

This work focuses on the transmission of high quality real-time multimedia within home and automotive environments. For this purpose the following QoS parameters are the most relevant ones on the system layer:

- Throughput
- Transit delay (latency)
- Transit delay variation (jitter)
- Error rate
- Packet loss

Audio/video data streams often have a fixed data rate, which is known in advance (in contrast, it is also possible that a stream adapts the data rate if the available bandwidth changes). Therefore, a certain throughput is necessary. In case of interactive two-way communication (e. g. video conferences) latency, i. e. the average transit delay, should be below a certain limit, typically fractions of a second. Otherwise, irritations of the communication partners are to be expected because the idle times between questions and responses from the other side become too high and the probability of both (or several) parties talking at the same time gets higher because it takes too much time to detect such collisions. However, for one-way transmission latency is usually no problem. The transit delay variation (jitter) should be limited. The limit depends on the buffer used by the sink device to compensate the jitter. The other way round, a higher jitter makes bigger buffers at the sink device necessary and, consequently, increases the presentation delay introduced in the sink device. Error rate and packet loss should be low. Depending on the format of the data stream, it might be more or less sensitive concerning data corruption. Detailed definitions of various QoS parameters for multimedia systems are given in [4].

If the system can guarantee the required QoS, the service will be called *guaranteed service*. If no QoS parameters can be guaranteed and the system just tries to deliver data as well as possible the service will be called *best effort service*. Most data networks (like the Internet to mention the most prominent example[1]) only provide best effort service. There are also other service classes defined, e. g. *predictive services*, which provide very soft guarantees [3]. For real-time streaming guaranteed services are desirable. In case of best effort services, real-time streaming can be performed, but disruptions are to be expected in case these requirements are not met all the time. Whether this is acceptable depends on the required perception QoS on the user layer.

The most important features a network should provide in order to enable smooth streaming of multimedia are summed up under the term isochronism. It is not an additional feature to the above-mentioned QoS parameters, but it is more a combination of several QoS parameters. It can be seen as a third characteristic to asynchronous data transfers, where there are no timing

[1] In some parts of the Internet, certain QoS is guaranteed, but in general, the Internet does not guarantee QoS.

constraints at all and synchronous data transfers, which mean according to [3] that there is an upper bound for the transit delay (but no limitation on the jitter). According to [4], an end to end network connection is called isochronous if the bit rate is guaranteed and the jitter is low and guaranteed as well. Another author defines isochronous data transfers as transfers where there is an upper as well as a lower bound for the transit delay, which, in fact, means that the jitter is limited [3]. In this work the definition of isochronous data transfers according to IEEE 1394 is preferred [19], where "isochronous" is defined as follows:

> "The essential characteristic of a time-scale or a signal such that the time intervals between consecutive significant instances either have the same duration or durations that are integral multiples of the shortest duration."

This definition, of course, matches the implementation of the isochronous data transfer on IEEE 1394, where isochronous data is sent in regular intervals, every 125 µs.

2.2 Resource Management

In order to guarantee a certain level of QoS, some resources are involved. Therefore, a system usually has to employ some kind of resource management. Other kinds of network management, which are not so tightly related to the provision of QoS, are not covered here. A communication system consists of many resources, sometimes they are classified as active and passive ones. Active resources are, for instance, a CPU or a network adapter that actively perform operations. Passive resources are file systems, transmission media, etc. However, this distinction is not always clear. For example, a network repeater can be seen as an active or passive resource. On the physical layer it performs operations on the signal, but from a higher layer's point of view it only relays data from its input to its output without any modification. Another classification of resources divides them between shared and exclusive ones. The latter can only be used by one process at a time, whereas the former can be used by several processes simultaneously. Most active resources are exclusive ones and most passive resources are shared ones.

Since certain resources are necessary to provide a certain level of QoS, a process requiring this level of QoS has to reserve these resources in order to make sure that they will be available for its purposes. There are many different kinds of resource management systems. There are systems on which the sink device or application is in charge for initialising the reservation ("receiver oriented protocols", e. g. RSVP) and others on which this is the duty of the source ("sender oriented protocols", e. g. ST-II). Although literature (which is often focused on IP based networks) usually only mentions sender oriented and receiver oriented reservation mechanisms, there is also a third kind: There are systems on which the application in charge of reserving resources may reside on a device independent from source or sink (e. g IEC 61883-1 used on IEEE 1394). A classification of QoS negotiation mechanisms (including resource allocation) is given in [3].

Another distinction between resource management systems is the degree of enforcement of the reservations. There are resource management systems that just manage reservation information without actually enforcing that a device or application does not utilise more resources than it has reserved. These systems, which the isochronous resource manager on IEEE 1394 buses is one example of, rely on the fact that the applications only use resources they have reserved. If some application does not use the reservation mechanism and utilises resources it has not reserved, it might break the whole resource management system because there might not be enough

remaining capacity of the resources for applications which have reserved correctly. Thus, such systems only work well if all devices and applications obey the reservation rules, but buggy or compromised devices can make the whole system fail. If a resource management system should enforce that not more than the reserved amount of resources is used by a specific application or data flow, it has to keep track of the resources reserved for each data flow. Resource management systems that only track the overall reserved or free respectively resources, but do not map reserved resources to specific data flows cannot enforce that each data flow is restricted to the resources reserved for it. Most systems—even if they maintain a mapping between data flows and reserved resources—do not actively enforce the correct use of the resource management procedures. Therefore, it is important that each application or device obeys to the rules of the resource management system involved. Otherwise, these systems would not be effective and QoS cannot be guaranteed.

Resource management systems also differ in the way the required resources are defined. There are systems that require applications to reserve the highest amount of resources they will ever need. This effectively prevents resource conflicts and is the only way to provide hard guarantees. The disadvantage of this approach (which is called "pessimistic approach" by [3]) is that resource utilisation might be suboptimal, especially in case of unsteady, bursty streams. The other way ("optimistic approach") means that the average of the required resources is reserved. Therefore, the utilisation of resources is optimal at the price that there might be situations when the needed resources are not available. Real systems usually constitute a mixture of both approaches. It is also possible that applications specify two values in the reservation process, the maximum value and a save value, which lies between the maximum and the average. For multimedia streaming applications the "optimistic approach", which can only provide soft guarantees, is usually sufficient.

To ensure that a data flow does not exceed its resource limits, two mechanisms can be involved to adapt a stream that does not fulfil the requirements by itself: traffic shaping and scaling. Traffic shaping can be used to "smooth" a data flow. Flows with unsteady traffic are smoothed (by delaying some data in case of bursts) to show a steady data rate that is lower than the bursts of the original stream. Traffic shaping does not modify the data itself, it just adapts the timing of transmission. There are a few mechanisms for traffic shaping [3, 9], two important ones are:

Leaky bucket is the name of an algorithm where the data to be sent is stored in a queue, the bucket. This bucket drains with a certain rate, which limits the maximum outgoing data rate. The size of the bucket limits the amount of data that can accumulate. If more data arrives, the bucket will overflow, and data will be discarded. This bucket size also limits the maximum delay caused by the bucket.

Token bucket refers to another mechanism where for for transmission of a packet a certain number of tokens is necessary. These tokens accumulate in a bucket, the token bucket, with a certain rate and are taken from the bucket when a packet is to be transmitted. If there are not enough tokens in the bucket for transmission of a packet, it has to be delayed. The size of the bucket limits the allowed bursts of data transmission, the token rate designates maximum long-term data rate.

There are other traffic shaping mechanisms as well, in particular variants and combinations of the above-mentioned ones [3].

Scaling means that the stream is adapted to fit to the QoS available. Scaling can be performed directly at the source of the stream, but it can also be done on intermediate systems on the way to the sink(s). This usually happens on the device where the available QoS degrades to a level impeding transmission of the original stream. There are two kinds of scaling:

Transparent scaling means modification of the stream without interaction with the higher layers. For example, the system layer can, if it detects that it cannot provide enough throughput to transport an MPEG-2 video stream, first drop B frames, which no other frames depend on, and then drop P frames, which only B frames and other P frames depend on. A description of the different frame types is given in Subsection 3.1.7. This, of course, requires that the system layer can detect the start and kind of frames within the stream. Modern data formats often provide such information that makes transparent scaling easy and define special profiles for scalable streams.

Non-transparent scaling requires interaction with higher layers. For example the stream is transcoded to another data format (e. g. from MPEG-2 to MPEG-4) or re-encoded with other encoding parameters (resolution, sample rate etc.).

2.3 Multicast

Since multimedia data usually has rather high bandwidth requirements, the available network bandwidth should be used economically. A major issue is that if there is more than one sink for a stream within a network, it should be avoided to transmit the stream several times and, therefore, utilise a multiple of the necessary bandwidth. Some networks only support unicast transmission that is directed to one specific sink. In this case, one connection for each sink is required leading to a waste of bandwidth.

For small networks in the home a possible solution could be the use of broadcast transmission, i. e. transmission targeting all nodes in the network. The major drawbacks thereof are the fact that also nodes not interested in the stream have to handle it and, in case of networks consisting of more segments, the bandwidth is used on all segments, even on those without an interested sink device.

The ideal solution is multicast, transmission to a group of targets. This is usually done by use of special "group addresses". In this case the stream is transmitted exactly once on the network segments needed. In segmented networks the coupling units between them (usually routers or bridges) need some additional "intelligence" to figure out, on which segments the data has to be replicated. There are dedicated protocols for devices to negotiate whether they want to receive a particular stream.

Transmission that is targeted towards more than one device is usually not reliable. The data is transmitted once and there is no retransmission in case of errors. There are several reasons, why there is no retransmission in case of error:

- In order to detect transmission errors, some kind of acknowledgement from the receivers would be necessary. In case of positive acknowledgements this would—especially in case of large multicast groups—lead to performance problems at the sender because it would receive very many acknowledgements ("acknowledgement implosion"). Furthermore, it would require that the sender knows all members of the group, which is not always possible. This can be solved partly by the use of negative acknowledgements, but this also has problems [20].

- The sender can either send the repeated data again to the complete group increasing the network traffic at every single node, or it can send it using unicast transmissions to each node that did not correctly receive it and, therefore, waste bandwidth by sending the same packet multiple times.
- Since usually the sources maintain the order of the packets to send, the repetition of data not sent successfully to other nodes would delay the following data units. This is not acceptable for real-time data streams.

This exactly matches the requirements of streaming: An erroneous packet should not be retransmitted because it would then probably be too late and, therefore, useless. The source should rather try to send the following packets in time than dealing with the erroneous packets. Timing is much more important than data integrity. The sinks must be able to cope with a certain (limited) amount of errors. There are applications that require reliable multicast communication, which have to deal with the problems mentioned above, but for real-time streaming of multimedia data, it is not necessary to require absolute reliability.

2.4 Environmental Requirements

The requirements mentioned above deal with the communication properties of the various networks. For the sake of completeness, it has to be mentioned that the networks used for multimedia streaming have to fulfil requirements depending on the environment they are used in.

One criteria is the allowed spatial dimension of the network, in particular the allowed length of cable segments. For automotive environments, several meters are the lengths which are used. In home environments it depends, whether the network is only used for connection within one room or within the whole flat or house. In the first case, several meters might be the required distances, whereas in the latter case several tens of meters or even hundreds of meters might be necessary.

The communication media itself has to fulfil certain requirements. For example, in automotive environments the range of the operation temperature that has to be supported and the allowed vibrations are much higher than in home environments. Other properties like cable thickness, bend radius and EMC issues (electromagnetic compatibility) depend very much on the particular environment. All these constraints have to be considered when a network is be installed, but they are not evaluated here in detail.

3 State of the Art

Distribution of multimedia data over networks is no new topic. There are several solutions—open as well as proprietary ones—to transmit various types of multimedia over different kinds of network. Here an introduction is given into the kinds of data to transmit, their data formats and the networks used for this purpose.

3.1 Data Types and Formats

The two major classes of multimedia data that are covered in this work are audio and video. Other kinds of multimedia data like text, animations, etc. are not of interest here. Both are created by sampling and quantification of an analogue signal in time domain. Then some kind of signal processing and compression is applied depending on requirements put on the created stream. There is a vast number of different data formats, so it is almost impossible to cover all of them. Here a short overview over some important ones is given, but this can by no means be complete. The focus is on formats that are used for streaming of multimedia. This list contains the most universal data formats, which are the raw data formats obtained directly after conversion from analogue to digital before compression algorithms are applied. Furthermore, the most important compressed data formats are explained, in particular the ones which are commonly used on IEEE 1394 and IP based networks, two typical networks used for multimedia streaming in home environments. The data formats are not explained in detail, but the main characteristics are described, and in particular, the properties with regard to real-time streaming are lined out.

3.1.1 Audio PCM

The simplest form of digital audio is PCM (pulse code modulation) coded audio [21]. This means that the audio is sampled in the time domain. For telephone quality audio 8000 samples per second are used, for CD-DA (compact disc digital audio) 44100 samples per second, and DVDs (digital versatile disc) or DVB (digital video broadcasting) usually use 48000 samples per second. According to Shannon's and Nyquist's sample theorem, the half of the sample rate determines the highest frequency within the audio signal that can be reproduced from the digital signal [22]. Therefore, a PCM stream with 41000 Hz sample rate contains frequencies up to 20500 Hz of the analogue audio signal. This perfectly matches the frequency range humans are able to hear. So the former continuous time signal becomes a discrete sequence of amplitude values.

The second step is the quantisation of the continuous amplitude values. This quantisation can either be linear, where the steps between the available amplitude levels are equal, or non-linear, where usually a logarithmic scale is used with finer steps for low signals and wider steps at the high signal levels. This correlates well to the perception characteristic of the human ear, which has a finer resolution at low signal level. Furthermore, speech usually contains more low amplitudes, and consequently, the logarithmic scale provides a better signal to noise

ratio (SNR) than linear quantisation. Therefore, logarithmic quantisation requires less bits per sample than linear quantisation for the same perceived audio quality. In telephone systems a logarithmic scale is used (μ-law in USA, A-law in Europe, defined in [21]) with 8 bits per sample. Consumer electronics (like CD-DA, DVB, DVD etc.) use 16 bit linear quantisation, whereas for studio quality 24 bit per sample are employed.

After these two steps (sampling and quantisation) the former analogue (time and amplitude continuous) signal has been transformed into a digital signal (time and amplitude discrete). Beside sample rate and the number of quantisation levels a third parameter is the number of channels used. While telephone systems only use one channel (mono), consumer electronic often employs two (stereo) or more channels.

Using PCM each sample is coded independently from the samples before and afterwards. However, there exist also variants of PCM that take into account the history of samples. These variants use a prediction for each sample, usually the predicted amplitude for the current sample is the amplitude of the previous sample. Differential PCM (DPCM) codes the difference to the previous sample. Since the difference of successive samples is usually small, less bits per sample are needed. The disadvantage of this method is that very fast amplitude changes might not be coded correctly. A slightly improved variant of DPCM is adaptive DPCM (ADPCM) which can adapt the step size used for quantisation of the difference to the previous sample. This allows better tracking of fast amplitude changes.

An extreme variant of DPCM is delta modulation (DM), which only uses one bit for coding of the difference to the previous sample. So basically each value tells if the amplitude is greater or less than the previous sample. This protocol requires very high sample rates to track fast amplitude changes. There is also an adaptive variant (ADM), which allows adaptation of the step size.

PCM audio, which can also be referred to as uncompressed audio samples, is the starting point for all compression mechanisms used. These compression techniques can roughly be classified into two classes:

Entropy Coding does not take into account the type of media, but only treats the stream as a sequence of digital data. Examples are statistical coding like Huffman Coding or Shannon Fano Coding, which use the fact that symbols usually occur with different probabilities, or Run Length Encoding, which take advantage of the fact that there are often sequences of the same data value. These coding methods are always lossless, i. e. after encoding and decoding the bit stream is exactly preserved.

Source Coding takes into account the type of media (e. g. music, speech, video, etc.) and uses the special properties of the media to achieve good compression. These methods can either be lossless or lossy. In the latter case an encoding and decoding cycle does not preserve the bits exactly, but leads to a data stream which is just more or less similar to the original one.

3.1.2 FLAC and Other Lossless Audio Codecs

FLAC stands for “free lossless audio codec”. This format is not of particular interest with respect to multimedia streaming (although it can be used for this purpose), but it is an example for a lossless audio compression method. Unlike other lossless compression technologies like the Lempel-Ziv-Welch (LZW) algorithm it is specially designed to achieve good compression on audio signals. Essentially it uses some prediction functions and stores the parameters of

these function together with the error signal. Lossless audio codecs achieve compression ratios around 1:2. More information about lossless audio compression can be found in [23]. However, multimedia streaming applications usually do not require lossless transmission because no exact copies of the original signal are needed, but a certain audio quality is desired, no matter how the signal is transformed. Therefore, if compression is used, usually the more efficient lossy techniques will be applied.

3.1.3 CELP, Speex, GSM and Other Speech Codecs

For transmission of speech at low bit rates special compression algorithms have been designed, which use the special properties of speech to achieve high compression ratios. Since these compression algorithms model the human voice they are not suitable for other audio signals like music. They are always lossy. CELP (code excited linear prediction) is one example for such technique. Speex is a codec based on CELP and pays much attention to avoid any patented methods. As the cellular telephone system GSM (global system for mobile communications) has its own speech codec, the main application of these speech codecs becomes clear: they are mainly used in cellular phone systems as well as in voice-over-IP telephone systems. Using such special codecs it is possible to transmit intelligible speech at around 5 kbit/s or even below. At very low data rates the speech can sound artificial and very different to the voice of the original speaker.

3.1.4 MPEG Audio and Vorbis

While PCM audio can be referred to as uncompressed audio and DPCM (and its variants) can be seen as a usually lossless compression (with some restrictions, in case of higher frequencies it becomes lossy), MPEG employs lossy compression [24]. As in PCM several audio channels are sampled using some sample rate. They are fed into a filterbank, where 32 bandpass signals are generated. These 32 subband signals are subsampled using 1/32 of the original sample rate. A psychoacoustic model is employed to decide about the resolution of quantisation required in order to keep the noise introduced by quantisation low enough. This psychoacoustic model takes into account various masking effects. For instance, if there is one loud tone at a certain frequency, the human ear will not be able to hear a sound much more quiet at a close frequency. This is called frequency masking. So a loud sound masks near frequencies, and therefore, quiet sounds at near frequencies are irrelevant and need not be coded. In other words, at frequencies close to loud sounds a higher noise level is acceptable. Another kind of masking is temporal masking. Loud sounds mask soft sounds (at close frequencies) for some time after the loud sound (up to 100 ms, this is called postmasking), and even a short period (around 5 ms) before the loud sound (premasking) soft sounds will not be heard. So the psychoacoustic model is used to allocate the available bits (according to the target bit rate) to the individual subbands. If the signal energy of a subband is too low, it will not be coded at all. That way, signal components that are not heard are removed and additional noise is introduced if it is not heard. In contrast to lossless compression algorithms, which only reduce redundancy from the data stream, lossy algorithms like the ones used by MPEG also remove irrelevant information. At the end of the MPEG audio coding process a (lossless) Huffman coding is applied.

There are some numbers used to mark different versions of the MPEG audio standards. First, arabic numbers are used to identify different phases of MPEG (MPEG-1, MPEG-2, MPEG-4, MPEG-7). MPEG-1 audio [25] provides three layers, usually numbered with roman

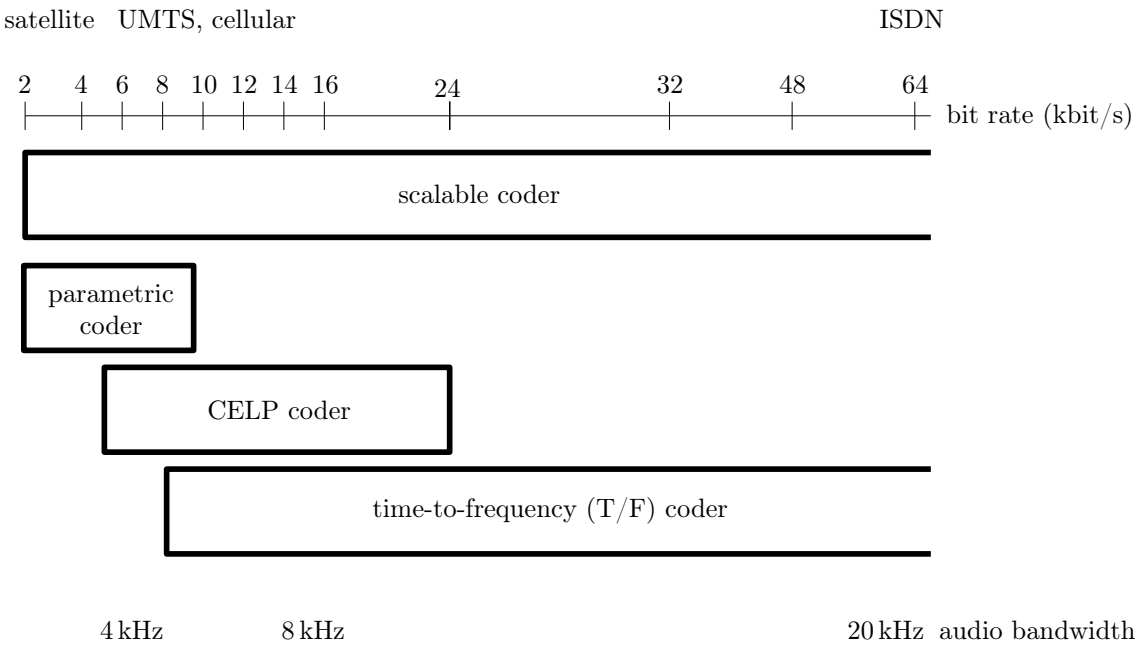

Figure 3.1: General framework of MPEG-4 audio

figures, layer I, layer II, layer III. The layers basically differ in the complexity of the encoder and decoder, each layer adds some functionality to increase the coding efficiency. Sometimes MPEG-1 layer II is referred to as MUSICAM. This is incorrect. MUSICAM was an older audio coding system, which was used as basis for MPEG-1 audio. MPEG-2 audio [26] is a backwards compatible extension to MPEG-1 audio, which adds additional supported sample rates, extends the bit rate range and supports more channels, up to 5 main channels plus a low frequency enhancement (LFE) channel, whereas MPEG-1 audio at most supports two channels (stereo sound). MPEG-1 audio and MPEG-2 audio can provide a data reduction rate of 1:4 (layer I) up to around 1:10 (layer III) at good quality. MPEG-2 AAC (advanced audio coding) [27] is a very high-quality coding standard supporting up to 48 main channels, 16 LFE channels, sample rates from 8 to 96 kHz, etc. It is not backwards compatible to MPEG-1 audio and it provides three profiles with varying levels of complexity and scalability.

The target for MPEG-4 development was streaming at very low bit rates. Therefore, MPEG-4 audio [28] introduced some compression techniques allowing very high compression ratios. The main thing is that in MPEG-4 the audio signal is divided into several audio objects, which are coded independently of each other. These objects can be natural or synthetic ones as speech, natural music, synthetic music, synthesised speech, etc. And for each audio object a suitable compression algorithm can be used like the ones known from MPEG-2 AAC, parametric coding techniques, CELP based coders as used for speech compression, text-to-speech decoders, etc. as shown in Figure 3.1. Another important feature of MPEG-4 audio is scalability of the bit rate of an audio bitstream and of encoder or decoder complexity.

MPEG-7 audio is a standard to provide meta-data about audio streams. It does not address compression of audio signals, and therefore, it is only of very little use concerning multimedia streaming.

MPEG only specifies the syntax and semantics of the stream and the operation of the de-

coder. It does not specify how an encoder works. This is up to the implementation and allows improvements of the psychoacoustic model and the encoding mechanism without modification of the standard.

Vorbis is an audio codec defined by the Xiph.org Foundation. The focus of the Xiph.org Foundation is to define codecs that are completely free, open, and unpatented. So Vorbis is not encumbered by patents and, thus, may be implemented freely by any individual or organisation. It offers similar features, e. g. compression ratios and audio quality, like MPEG audio and uses similar mechanisms, for example a psychoacoustic model. It is often used as an alternative to MP3 (MPEG-1 layer III).

3.1.5 Video Uncompressed

As in case of audio, the two steps to transform video from analogue to digital are sampling and quantisation. But in difference to audio, video is not only sampled in the time domain, but also in the spatial domain. The minimal picture frequency for flicker-free video is around 50 Hz [3]. Systems that cannot provide 50 full pictures per second often use interlacing to provide "half frames" (called fields), one field consisting of odd lines and one field consisting of even lines. The field rate is the double of the frame rate. If the frame rate is high enough, progressive scan will usually be preferred, since interlace mode can cause various problems (especially when it has to be transformed to progressive mode) and at last is just an auxiliary means to overcome the flicker caused by a too poor frame rate [29].

For spatial sampling the picture is organised as a (rectangular) raster of rows and columns of picture elements (pixels). Each pixel gets some colour information assigned. This can either be just a luminance value in case of monochrome images or a colour value in some colour system. When it comes to displaying the video (i. e. the pictures of the video), e. g. on a CRT or using a projector, usually the RBG (red, green, blue) colour system is used, but for transmission or storage of video usually other colour spaces are employed, for example YUV for PAL TV (phase alternating line, the analogue TV standard used in most European countries) systems and YIQ or YUV for NTSC TV (National Television System Committee, defines the analogue television standard used in the USA). In both colour spaces, the Y component stands for chrominance, while the other ones (UV and IQ, respectively) are used to store chrominance. In colour spaces providing a separation between luminance and chrominance information, many systems use a higher resolution for luminance information than for chrominance. This matches the properties of the human eye, which is more sensitive to luminance than chrominance differences. It could be noted that all variants of digital images do not reproduce the original colour of the object by recording the spectrum of the electromagnetic waves (within the optical range) and reproducing it. In fact, the colours are generated by mixing of some primary colours so that the mixture appears to be the same for a human observer (this means generating the same stimulus at the neurons in the human eye, which are the cones in this case). In this respect, storing of colour information can always be seen as a kind of lossy compression because the original electromagnetic signal cannot be recovered. The resulting spectrum might be very different from the original one, but generates the same (or similar) impression for a human eye.

For the purpose of storing digital video, the CCIR-601 video standard [30] defines a stream of 25 fps (frames per second) in PAL (30 fps for NTSC), 625 lines (525 for NTSC) and 864 samples per line (858 for NTSC, in both cases 720 samples are active) as studio quality video. This leads to a sample rate of 13.5 MHz. If the luminance signal and both chrominance signals were quantised with 8 bits, this would result in 324 Mbit/s. But since the active area is only 720×576

for PAL (720×480 for NTSC[1]), about 249 Mbit/s are needed for studio quality PAL video. If the two chrominance signals are subsampled with a factor 1:2 approximately 166 Mbit/s will be sufficient. This is denoted as 4:2:2 subsampling, where two samples of both chrominance components come with 4 samples of luminance in each row. If the chrominance signals are subsampled with a factor 1:4 (video conference quality), 124 Mbit/s will be needed. This can be performed either by 4:1:1 subsampling, where one sample of each chrominance component is stored per 4 luminance samples per line (the horizontal chrominance resolution is quartered, the vertical resolution is unchanged), or 4:2:0 subsampling, where 2 samples of one chrominance component and no sample of the other one are stored per 4 luminance samples changing line by line (both horizontal and vertical chrominance resolutions are halved).

Reducing both the vertical and horizontal resolutions of CCIR-601 digital video by a factor two leads to the source input format (SIF, sometimes also referred to as standard interchange format) requiring a quarter of the bandwidth (about 31 Mbit/s). This turns former interlaced video into progressive one. It results in 360×288 at 25 Hz (PAL) or 360×240 at 30 Hz (NTSC). Since 360 is no multiple of 16 (which is desirable for many coding mechanisms), it is often "rounded" to 352 leading to 352×240 (NTSC) and 352×288 (PAL). Unfortunately, SIF is only mentioned in an informational (non-normative) annex of [31] and slightly different dimensions are commonly used. Even inside this annex the exact dimensions of the SIF format are not consistent[2].

CIF (common interchange format) is defined in ITU Recommendation H.261 [32], see Subsection 3.1.8. It always uses 352×288 pixels at 30 Hz. There is also a variant called QCIF (quarter CIF) using 176×144 pixels.

It should be mentioned that many of the above-mentioned formats do not use exactly square pixels. Since the picture aspect ratio (width:height) is defined as 4:3 for these formats, it can be seen easily that for some formats the pixels are slightly wider than tall while for others they are slightly taller than wide.

3.1.6 M-JPEG, DV

Motion-JPEG (M-JPEG) is video data consisting of a sequence of JPEG compressed images. While JPEG is well standardised [33], there is no standard for M-JPEG and, therefore, implementations of different vendors are often not interoperable. JPEG uses DCT (discrete cosine transformation) for a lossy compression; there is a trade-off between quality and compression rate. Therefore, the compression rate and, in consequence, the data rate of M-JPEG is tunable in exchange for quality.

DV stands for digital video and is a data format specified in [34–36]. It was developed more or less in parallel to IEEE 1394 and was designed to be perfectly suited for transmission over IEEE 1394. In fact, the structure of the data on the tape is very similar to the packets transferred over IEEE 1394 as specified in [37–39]. It is typically used on digital camcorders and VCRs. Although it is called DV (digital video), the data stream can also contain audio, and therefore, it could be regarded as a multiplex streaming format. The data rate for SD format (standard definition, which is the CCIR-601 resolution: 720×480 at 30 fps or 720×575 at 25 fps) video is 25 Mbit/s; together with audio and subcode data it yields about 29 Mbit/s. Audio is

[1] In some literature, an active area for NTSC video of 720×484 or slightly different dimensions is mentioned. So the vertical size of the active area is not always used consistently.

[2] The NTSC version is at one place specified as 360×240 and at another place as 360×242. However, if rounded to dimensions divisible by 16, the result would anyway be 352×240.

encoded either as two channels at 48 kHz sample rate and 16 bit resolution or as four channels at 32 kHz sample rate and 12 bit resolution, in both cases leading to approximately 1.5 Mbit/s data rate. On the tape it even requires even around 36 Mbit/s because error detection and error correction codes are added. For video encoding either 4:1:1 (NTSC version) or 4:2:0 (PAL version) subsampling of the chrominance components is used. Therefore, the video compression ratio compared to 124 Mbit/s for uncompressed video is around 1:5.

An advantage of these formats (M-JPEG and DV) is the low complexity of encoding and decoding because it does not use the sophisticated inter-frame compression techniques known from formats like MPEG-1 and MPEG-2. On the other hand, this imposes rather high bandwidth requirements. Furthermore, they allow frame exact cutting because there is no dependency between frames and, therefore, are ideal for editing.

Like M-JPEG DV is also based on DCT, but since it uses more local optimisation, it produces higher quality at the same data rate. DV provides very high video quality in the area of consumer electronic at the expense of rather high bandwidth requirements.

There are other similar formats, e. g. digital betacam, which are targeting more the professional than the consumer electronic area. Since this work is directed to home environments rather than to video studio environments, they are not investigated further here.

3.1.7 MPEG Compressed Video

MPEG is a family of very sophisticated video compression methods [24]. The older version MPEG-1 video [31] was extended by its successor MPEG-2 video [40]. It is a lossy compression algorithm using DCT for spatial redundancy reduction and motion compensation for temporal redundancy reduction. It uses three kinds of pictures[3]:

I (intra-coded) pictures: These pictures are completely self-contained and do not use any reference to other frames. They provide access points where decoding can start. The compression is rather moderate at typically around 1:7.

P (predictive-coded) pictures: These pictures use motion compensation to reduce temporal redundancies. They contain references to the previous I or P picture. They can be compressed significantly better than I pictures, the typical compression ratio is around 1:20.

B (bidirectionally predictive-coded) pictures: These pictures use references to the previous I or P picture as well as to the next I or P picture. They can either use references to the previous or to the next picture (forward or backward motion compensation) or to both (interpolation). B pictures are never used as references for other pictures and, therefore, do not contribute to error propagation. They yield the highest compression ratio around 1:50.

Video streams containing B pictures require that the transmission order is different to the display order leading to a higher overall delay. Furthermore, they require more memory in the decoder as two reference pictures have to be stored. It is up to the encoder whether B pictures or even P pictures are used and in what ratio the various types of pictures are used. This is mainly a trade-off between compression ratio and the number of possible access points.

[3] Depending on whether video is progressive or interlaced, the term picture may either refer to a frame or to a field.

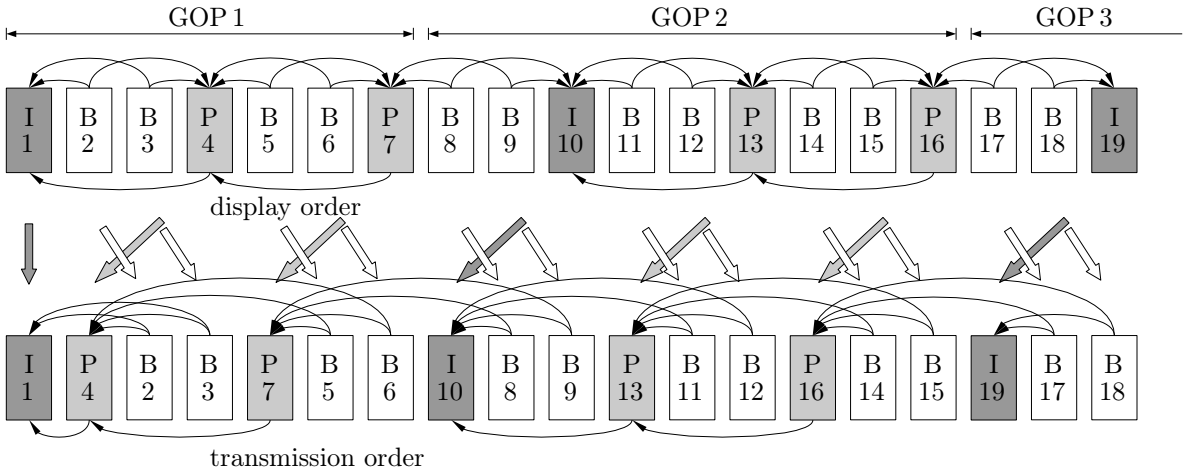

Figure 3.2: Structure of GOPs (groups of pictures) in MPEG-2 video streams

Therefore, the overall compression ratio can vary significantly, but it is typically between 1:10 and 1:30.

MPEG-1 video is typically used for applications at 1.5 Mbit/s and SIF (see Subsection 3.1.5) video format, which is typically found on Compact Discs although it allows higher data rates and resolutions (4095×4095 and bit rates up to 100 Mbit/s).

The mechanisms used in MPEG-2 are similar to those known from MPEG-1. The main improvement is the support of interlaced video in addition to progressive video as well as some improvements to coding efficiency. Typical applications for MPEG-2 are digital television (e. g. DVB [41]) and movies on DVD.

The pictures in MPEG video are grouped to groups of pictures (GOP), which contain at least one I picture and, therefore, a starting point for decoding. The length of the GOPs is an important coding parameter. For editing, usually rather short GOPs are used, whereas for broadcast purposes longer GOPs are common. Figure 3.2 shows a typical structure of GOPs in MPEG-2 video streams. In this figure, each picture is labelled with its type (I, P or B) and its sequence number in display order. The references to other pictures for motion compensation are denoted with arrows. It visualises the dependencies between pictures and the necessity to reorder the pictures before transmission in order to create a stream that only contains references to previously transmitted pictures.

MPEG-2 defines profiles and levels, where the profiles (Table 3.1) limit the syntax and algorithms used, they determine bitstream scalability and colour space resolution, whereas the level determines the coding parameters like sample rates, frame dimensions, etc. For example, the simple profile does not use bidirectional prediction. Therefore, no B pictures are used in this profile. The most important combination is main profile and main level (abbreviated MP@ML). Main level (ML) stands for CCIR 601 (see Subsection 3.1.5) resolution, whereas low level (LL) refers to SIF resolution (see Subsection 3.1.5), high level (HL) is used for HDTV resolutions. All valid combinations of profiles and levels are shown in Table 3.2. This table also shows characteristics of these combinations like supported subsampling of chrominance components (see Subsection 3.1.5), maximum dimensions, maximum supported bit rate, and used picture types.

Profile	Comments
Simple	Only I and P pictures, no B pictures, intended for software applications and cable TV
Main	Most decoder chips, cable and satellite TV, 95 % of users
SNR Scalable	Scalable coding
Spatial Scalable	Scalable coding
High	Higher sample precision, 4:2:2 subsampling ratio in addition to 4:2:0 of the other profiles (see Subsection 3.1.5)
4:2:2	Later introduced profile for studio applications

Table 3.1: Characteristics of MPEG-2 profiles

Profile / Level	Simple	Main	SNR	Spatial	High	4:2:2
High	illegal	4:2:0 1920×1152 80 Mbit/s I, P, B	illegal	illegal	4:2:0, 4:2:2 1920×1152 100 Mbit/s I, P, B	illegal
High - 1440	illegal	4:2:0 1440×1152 60 Mbit/s I, P, B	illegal	4:2:0 1440×1152 60 Mbit/s I, P, B	4:2:0, 4:2:2 1440×1152 80 Mbit/s I, P, B	illegal
Main	4:2:0 720×576 15 Mbit/s I, P	4:2:0 720×576 15 Mbit/s I, P, B	4:2:0 720×576 15 Mbit/s I, P, B	illegal	4:2:0, 4:2:2 720×576 20 Mbit/s I, P, B	4:2:2 720×608 50 Mbit/s I, P, B
Low	illegal	4:2:0 352×288 4 Mbit/s I, P, B	4:2:0 352×288 4 Mbit/s I, P, B	illegal	illegal	illegal

Table 3.2: MPEG-2 profiles and levels

While MPEG-1 and MPEG-2 video are rather similar, MPEG-4 video introduced completely new concepts [42]. It is targeted towards lower bit rates and higher compression rates. The main difference is that in MPEG-4 the video can be split into several video objects which can be coded independently. For example, the still background can be coded using a lower bit rate, while a speaker in front can be coded with another algorithm and at a higher bit rate. Furthermore, there is better reusability of already transmitted content. It is possible to pan and zoom in a scene without the need for retransmission. Sprites can be used. It also includes mechanisms to allow interactions with the user. The possible compression rate is at least a factor 2 better than in MPEG-2.

The latest extension to MPEG-4, part 10, called “advanced video coding” (AVC) [43], can again enhance the compression with a factor 2 leading to compression ratios of up to 1:100. MPEG-4 part 10 AVC was developed jointly by the ISO/MPEG group and ITU-T as it was also done for MPEG-2 part 2 (video). A comparison between the modern compression mechanisms MPEG-4 ASP (advanced simple profile), MPEG-4 AVC and Microsoft VC-1 is given in [44].

3.1.8 ITU Recommendations H.261, H.262, H.263, H.264

These three formats have been defined by ITU-T [32, 45–47] and are intended for use in video conferences. H.261 is older than MPEG-1 and can be seen as a predecessor of MPEG-1 video. It uses very similar coding techniques and was designed for bit rates between 64 kbit/s and 2 Mbit/s[4] and picture formats CIF and QCIF. One remarkable difference is that H.261 does not support B pictures, it only uses I and P pictures.

H.262 is the same as MPEG-2 video. It was also jointly developed by ISO/MPEG and ITU-T.

H.263 was designed for video conferences at low bit rates. It is the successor of H.261 and providing many improvements and many similarities to MPEG-2 video. Important differences are, that H.263 does not support interlaced video, and MPEG-2 is more flexible concerning the resolution of video.

H.264 is the same as MPEG-4 part 10 (advanced video coding, AVC) [43]. It was jointly developed by ISO/MPEG and ITU-T.

3.1.9 Multiplex Streams

Besides data streams transporting one single audio or video signal, there are also some kinds of stream defined that can contain several video and audio streams at once. The MPEG standards define several types of multiplexed streams. While MPEG-1 defines one kind of multiplex stream, the MPEG-1 systems stream [48], the MPEG-2 standard provides two different kinds of multiplex streams [49]:

Program streams are used to combine video, audio and data streams of one program with a common time base into a single stream. The program stream is designed for almost error-free environments, the packets are of variable length and often rather long. They are typically used for storage media like DVD. MPEG-1 systems streams are similar, program streams can be seen as their extension.

Transport streams are designed for environments in which significant errors like bit errors or lost packets might occur. A fixed packet size of 188 bytes is used. Transport streams may consist of audio, video and data streams of several programs with one or more independent time bases. A typical application of Transport streams is digital television according to DVB (digital video broadcasting) or ATSC (Advanced Television Systems Committee) [41, 50, 51]. Here several TV programs are carried within a single transport stream.

An important feature of such multiplexed streams is the intra-stream synchronisation of several streams to ensure that audio and video are lip sync. For that purpose, audio and video elementary streams are packetised into packetised elementary streams (PES) which may contain decoding and presentation timestamps (DTS, PTS) within their headers. As shown in Figure 3.3, these PES are then combined to a program stream or transport stream. The stream also contains timestamps of one (program stream) or more (transport stream) reference clocks, which are called system clock reference (SCR) in program streams and program clock reference (PCR) in transport streams. All these timestamps together ensure synchronous playback of audio and video and avoid overflows and underflows of the data buffers of the decoders.

MPEG-4 defines a two layer multiplexer for synchronised delivery of multimedia. As in MPEG-2, elementary streams are transformed into (synchronisation layer) packetised elementary streams (SL-PES). The optional FlexMux (flexible multiplexing) tool can be used to

[4] To be exact: for multiples of 64 kbit/s, with a multiplier between 1 and 30

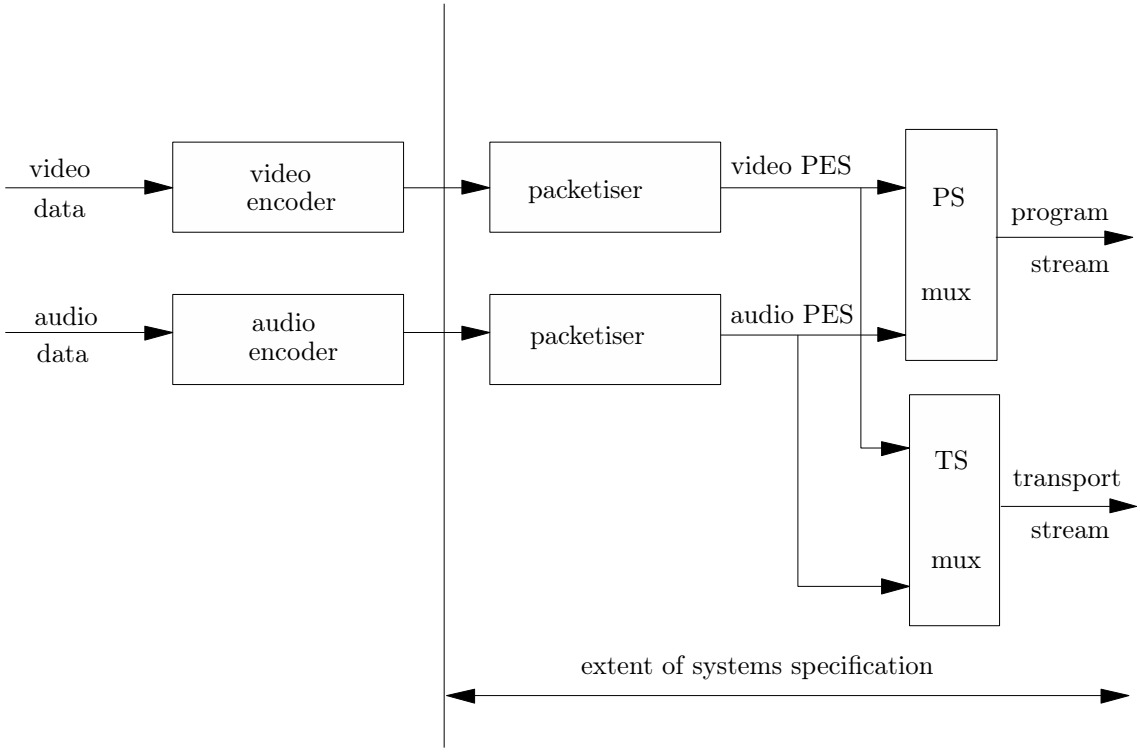

Figure 3.3: Overview over MPEG-2 systems [49]

combine one or more SL-PES to a FlexMux stream with little overhead. The TransMux (transport multiplexing) layer creates the final stream used for transport. MPEG-4 only specifies the interface to the TransMux layer. Many transport systems can be used to deliver MPEG-4 streams, for example RTP/UDP/IP or MPEG-2 transport streams. These transport systems have to be defined in other documents. Therefore, MPEG-4 maintains a very high flexibility to adopt many different transport protocols.

As already mentioned in the section about DV, this format can contain audio together with the video stream. Therefore, it also can be regarded as a multiplexed stream format.

There exist several container file formats that are used to store several multimedia within one file. AVI (audio video interleave) is a file format designed by Microsoft. Its main purpose is storage of multimedia data, but it was not intended for streaming. It is not suited for streaming because playback of an AVI file usually does not involve sequential reading of the file, but it requires seeking several location within the file to get all necessary information. The Quicktime file format developed by Apple also allows putting several media streams (they are called track here) into one file [52, 53]. The Quicktime file format uses a very flexible structure consisting of so-called atoms, which allows a hierarchical layout of multimedia data. Quicktime technology does not only consist of a file format, but it stands for a complete multimedia framework including a player, a streaming server, etc. Therefore, Quicktime is well suited for streaming. It allows interleaving of the data of several tracks, but it does not require to structure the data that way. This would be essential for streaming, if the Quicktime stream was streamed directly as is over the network. However, Quicktime movies are usually not streamed as Quicktime bitstream, but the tracks contained are streamed individually using

some transport protocol (like RTP, see Subsection 3.2.2). Therefore, interleaving of the data of the tracks is only necessary for progressive streaming, but not for real-time streaming, although it is advantageous. A file that is intended to be used for progressive streaming has to be created in a way that all meta data needed for decoding is placed at the beginning of the file. Quicktime allows more or less arbitrary order of atoms within the file. For real-time streaming, special hint tracks are required containing information for packetising the data for the transport protocol. In fact, the design of the file format for MPEG-4 is based on the Quicktime format. So there is a convergence between Quicktime and MPEG-4 technology.

Ogg is the container file format defined by the Xiph.org Foundation. Its main purpose is the storage of Vorbis bitstreams in files, but it can be used for different codecs and also allows multiplexing of several logical bitstreams. However, it is not intended for streaming, but only for storage. A Vorbis bitstream can be directly streamed over e. g. RTP without need for the Ogg container.

3.1.10 Proprietary Formats

There are several proprietary data formats for multimedia data—for audio, video and multiplexed data streams. One important group are the data formats designed by RealNetworks. RealNetworks was the first company working in the area of streaming multimedia in the Internet. They have been active since the middle of the 1990s (a first version of their RealPlayer was published in 1994). RealAudio stands for a set of different codecs for different purposes ("Voice", "Music", "Stereo Music") and different target bit rates, but the technical details are a secret. Since version 8, ATRAC 3 (adaptive transform acoustic coding) compression technology from Sony has been integrated enabling CD quality audio at bit rates up to 352 kbit/s. Former versions supported bit rates up to 80 or 96 kbit/s (depending on the exact version) leading to at most radio quality audio. RealVideo does not provide such set of different codecs, but there are just different versions of the RealVideo codec. Although the technical details are secret, it can be assumed that RealVideo as well as RealAudio use similar methods like MPEG-2 to achieve compression [53].

The second important group of proprietary data format are those from Microsoft. Since Microsoft started its activities in multimedia streaming very late, it had to use existing technologies to achieve a short time to the market. Therefore, Windows Media was strongly oriented towards MPEG, especially MPEG-4. The first versions of Windows media video implemented MPEG-4 following the recommendations of ISO very closely. More recent versions of Windows media video incorporate more proprietary technologies. Similarly, Windows media audio was inspired by MPEG-1 layer III (also known as MP3) and licensed codecs like Sipro Labs ACELP (algebraic code excited linear prediction) codec. Nevertheless, later versions introduced more proprietary technologies [53].

The third big commercial player in the multimedia market is Apple Computers with their Quicktime technology. Other than the above-mentioned companies, Apple did not develop its own codecs and designed Quicktime as an open format, which can take advantage of many different codecs, mostly open ones. But it also supports proprietary codecs like Sorenson video or the QDesign music audio codec.

3.2 Networks in Home and Automotive Environments

Typical networks in the home that are used for multimedia streaming are IEEE 1394 (also known as FireWire or i.Link) and IP based networks (usually Ethernet networks). In future automobile environments either IDB-1394 (the automotive version of IEEE 1394b, IDB stands for "intelligent data bus") or MOST (media oriented systems transport) will be used for multimedia in the car depending on the manufacturer. In fact, both IDB-1394 and MOST share many concepts, but also have fundamental differences [10, 11]. This section gives an introduction into networks used in home and automotive environments.

In this thesis, the term network is used to denote a communication system in which one transport protocol can be used to transport real-time multimedia data from one end device to another and in which one control protocol is used for the involved end devices to communicate with each other. If an intermediate system is needed between two parts of the communication system that has to intercept and interpret the transport or control protocol, these two parts will be seen as two separate networks. So even larger communication systems like the Internet, with routers connecting subsystems, are seen as one single network, if no intermediate system is needed that has to intercept and manipulate the control protocols or real-time multimedia streams a the layer of the transport protocol or higher.

3.2.1 IEEE 1394

IEEE 1394 [19], also known as FireWire or i.Link, is a serial bus designed to connect up to 63 consumer electronic devices (like camcorders, CD players, satellite tuners, etc.), but also personal computers and peripherals like hard disks, scanners etc. The original version of this bus supports up to 400 Mbit/s, has an automatic configuration mechanism allowing hot plugging of devices and is very well suited for streaming of multimedia content [8, 54]. It supports 4.5 m links and up to 16 hops between two devices leading to a maximum distance of 72 m between two nodes. This clearly shows, that it is suited for small area environments like homes or automobiles. IEEE 1394a [55] (which is the most commonly used version as time of writing) introduced some new features, mainly to increase the bus efficiency (reduction of idle times), but the main characteristics were preserved. The newer version IEEE 1394b supports up to 1600 Mbit/s and has architectural support for 3200 Mbit/s [56]. It introduced some new physical media in addition to STP (shielded twisted pair) copper cables and also increased the maximum distance between devices. Table 3.3 shows the different types of media and their supported data rates as well as the maximum distances between neighbour nodes in IEEE 1394b. The transmission model and coding of data is completely different to IEEE 1394a; however, compatibility is maintained by introduction of bilingual nodes.

There is a special version of IEEE 1394 adapted for automotive environments: IDB-1394 [10, 11, 57]. It is based on IEEE 1394b with some restrictions and additions. For the embedded network, POF (plastic optical fibre) is used as media, the other media defined by IEEE 1394b are not used. The maximum distance between two embedded nodes is limited to 10–18 m, depending on the number of inline connections. A special interface to connect portable devices to the bus is defined and called CCP (customer convenience port). This is a variant of the IEEE 1394b copper bilingual connector, so IEEE 1394b as well as IEEE 1394(a) devices can be connected to the embedded network. A major addition of IDB-1394 are special power management mechanisms to keep the power consumptions of parked cars low. For this purpose, ultra low power states for the ports were defined as well as a special power management protocol.

Media	Reach (m)	S100	S200	S400	S800	S1600
CAT-5 UTP (unshielded twisted pair)	100	X				
POF (plastic optical fibre)	50	X	X			
HPCF (hard polymer clad fibre)	100	X	X			
MMF (50 µm multimode fibre)	100			X	X	X
STP (shielded twisted pair)	4.5			X	X	X

Table 3.3: IEEE 1394b media

Figure 3.4 shows an example setup of an IEEE 1394 bus. It can be seen that on the physical layer it forms a tree consisting of point-to-point connections between the devices. However, the physical layer transforms this tree into a logical bus, where each node (except the node that is currently transmitting a packet) acts as repeater. The higher layers (link layer and transaction layer of the three-layer design used in IEEE 1394) have a view as a bus, where all packets are received by every node, provided the used speed is supported on all links in-between. In the example setup in Figure 3.4, the satellite receiver and the TV set can communicate with 400 Mbit/s (S400), but the camcorder does not receive these packets. Camcorder and TV set can communicate with S200 (200 Mbit/s) via the HiFi set; these packets will be seen by every node.

Two different data transfer modes are supported: asynchronous and isochronous data transfer. The isochronous mode has been designed to meet the requirements of streaming. There are 64 isochronous channels. Such channel can be regarded as a multicast group. In fact, the packets are transmitted to each node on the network (the restriction is that the physical layer of all nodes between the transmitting and the receiving node have to support at least the speed used for transmission of the packet), but the hardware inside the node usually filters and discards packets of isochronous channels that are not of interest. There is a resource management system: Bandwidth and channel can be reserved at the isochronous resource manager and are then guaranteed. This means that quality of service is provided. The requirement for the guarantee of isochronous bandwidth is that all nodes have to obey the rules of IEEE 1394 and reserve bandwidth and channel before they are used. Since this is not enforced by hardware or software, nodes violating these rules can compromise QoS. According to the classification made in Section 2.2, IEEE 1394 resource management uses a "pessimistic approach", where each single isochronous packet has to comply with the bandwidth reservations made. As already mentioned in Section 2.2, the node performing the reservation has neither to be the talker (source) nor a listener (sink) of the channel, but it might be a third node. Furthermore, the node performing the deallocation of resources, when they are no more needed, might be yet another node. The source can send one isochronous packet every 125 µs (leading to 8000 isochronous cycles per second). The allowed size of the packet corresponds to the reserved bandwidth. In case of errors, no retransmission occurs, so timing is guaranteed, but not delivery.

On the other side, the asynchronous transfer mode is used for delivery of variable-length packets to explicitly addressed nodes. There is an acknowledgement and a retry procedure in

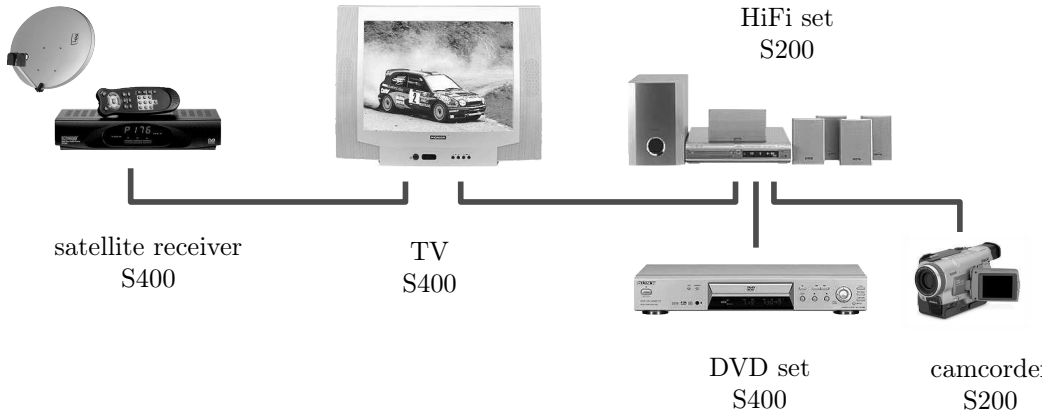

Figure 3.4: IEEE 1394 example setup

case of errors. Therefore, the delivery of integer data can be guaranteed, but timing is not guaranteed. For asynchronous transactions split transactions are used, so each transaction consists of a request subaction and a response subaction, which are independently acknowledged link layer data transfers and might be interleaved by other data transfers. Since there are nodes that would have a higher natural priority when arbitrating the bus, there is a fair arbitration mechanism leading to fair access to the bus without starvation of some nodes. In the area of streaming multimedia data, asynchronous transactions are not used for the transfer of multimedia, but they are used for control purposes. IEC 61883-1 uses asynchronous transactions for setup and teardown of isochronous connections which also includes the isochronous resource management [58]. The two major control protocols for consumer electronic multimedia devices, AV/C and HAVi, are based on IEC 61883-1 and use asynchronous transactions to communicate. AV/C (audio/video control) is a simple protocol to control single multimedia devices [59]. HAVi (home audio video interoperability) is a middleware architecture allowing more complex control as well as resource management and scheduling tasks [60].

Real-time transmission of audio and video data is well standardised by the series of IEC 61883. IEC 61883-1 defines general things like a common format for isochronous packets as well as mechanisms for establishing and breaking isochronous connections [58]. This also includes timestamps, which are used for intra-media synchronisation as well as for inter-media synchronisation [61]. The time base for these timestamps is the cycle time, which is synchronised between all nodes automatically by IEEE 1394. This is a very valuable feature of IEEE 1394: There is a clock that is automatically synchronised between all nodes and, therefore, does not drift. Consequently, this clock can be used and is used to lock the clocks of source and sink devices (e. g. in case of audio, the frequency of the DAC at the sink device can be locked to the ADC at the source device) to avoid buffer underruns and overruns.

Transmission of data in the DV (digital video) format [34–36] used on video cassettes (and, therefore, typically provided by VCRs and cameras) is specified in parts 2, 3, and 5 of IEC 61883 [37–39]. An advantage of this format is the high quality and a low complexity of encoding and decoding because it does not use the sophisticated inter-frame compression techniques known from MPEG-2. On the other hand, it imposes rather high bandwidth requirements (about 25 Mbit/s for an SD format DV video stream, together with audio and other information it

chunk length	number of chunks
14	5783
15	22728
16	0
17	0
18	381
19	1119

Table 3.4: Chunk length of full isochronous packets containing 480 bytes DV data between empty packets during one minute transmission according to IEC 61883-2

yields about 29 Mbit/s).

To distribute audio, IEC 61883-6 gets involved [62]. It is used by consumer electronic audio equipment like CD players, amplifiers, speaker systems, etc. Usually, audio is distributed as uncompressed audio samples at 44100 or 48000 Hz sample rate, but other data formats are supported as well.

Transmission of MPEG-2 transport streams [49] over IEEE 1394 is specified in IEC 61883-4 [63]. This allows the distribution of digital television over IEEE 1394, since both DVB and ATSC use MPEG-2 transport streams. MPEG-2 provides a good trade-off between quality and utilised bandwidth. A typical PAL TV program requires about 6 Mbit/s. [64] specifies how to distribute the content of a DVD over IEEE 1394 according to [62] and/or [63]. The alternate digital TV standard DirecTV/DSS can be distributed according to IEC 61883-7 [65]. So the ideal solution to distribute digital multimedia content is to use IEC 61883-6 for high quality audio data, whereas IEC 61883-4 can be used if there is also video to distribute.

Most isochronous streams transmitted according to IEC 61883 send data in every isochronous cycle. The size of the packets varies, the difference between the packet sizes is a multiple of the so-called data block size. So the current data rate always oscillates a bit around the average data rate. This is true e. g. for MPEG-2 transport streams according to IEC 61883-4 or audio data streams according to IEC 61883-6. Transmission of DV according to IEC 61883-2, IEC 61883-3 and IEC 61883-5 is a bit different because here the size of the packets is fixed (e. g. for SD video it is 480 bytes plus headers). DV data is structured in so-called DIF (digital interface format) blocks of 80 bytes. In each isochronous packet 6 DIF blocks are transported. 150 DIF blocks are one DIF sequence, so it needs 25 isochronous packets to transmit one DIF sequence. In NTSC, a frame consists of 10 DIF sequences and thus requires 250 isochronous packets. Since there are 8000 isochronous cycles per second, it would be possible to transmit 32 frames per second, although an NTSC video needs only 29.97 frames per second. A PAL/SECAM[5] frame consists of 12 DIF sequences resulting in 300 isochronous packets per frame. So it would be possible to transmit about 26.7 frames per second, whereas the video only needs 25 frames per second. The consequence is that during transmission of DV over IEEE 1394, there are also empty packets between full packets. According to above calculation, about every 16^{th} packet has to be empty. Experiments with a consumer DV camcorder available at the institute (Sony Digital Handycam DCR-PC100E) showed that during 1 minute transmission of DV video over IEEE 1394 the lengths of chunks of full packets between the empty ones were between 14 and 19 with the peak at 15 (see Table 3.4 and Figure 3.5). This means that even in case of a fix packet size as for DV transmission the actual data rate is not rigidly coupled to the isochronous

[5]SECAM is the analogue TV system used in France.

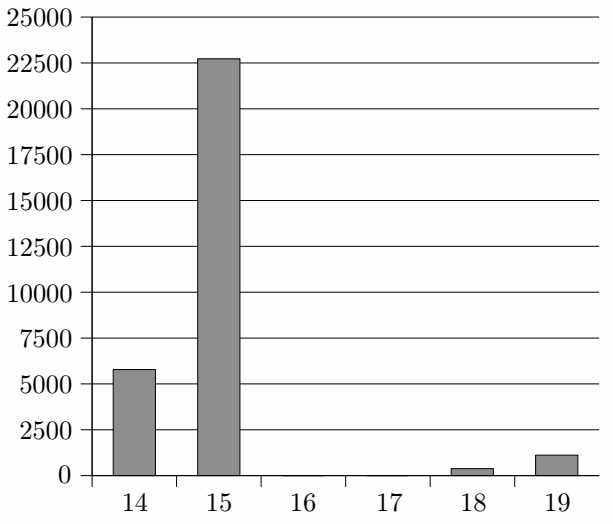

Figure 3.5: Distribution of lengths of chunks of full isochronous packets during one minute transmission of DV according to IEC 61883-2

cycle rate of the IEEE 1394 bus, but there is some flexibility with the maximum possible data rate about 6.7 % higher than the required nominal data rate. Therefore, although IEEE 1394 provides a clock used for scheduling data transmission, the timing of transported media (i. e. sample or frame rate) is independent of and not slaved to the IEEE 1394 cycle time. It is always possible to transport media at a rate slightly above or below the nominal rate. This is a difference to e. g. MOST, where the DAC for audio output is expected to directly use the bus clock.

IEEE 1394.1 is a standard for IEEE 1394 bridges [66]. It allows connection of up to 1023 IEEE 1394 buses (each containing up to 63 nodes) to a network. An advantage of those networks is that traffic tends to be isolated on one bus. This includes isochronous traffic and bus resets. Bandwidth is, therefore, used very economically. If there is a listener on another bus than the corresponding talker (IEEE 1394.1 uses this terminology), there is a special protocol to set up the bridges in-between to forward isochronous packets. Since the standard is rather new (it was approved in December 2004), no 1394.1 bridges are available as products at the time of writing. However, during the work for this dissertation a prototype of a (distributed) IEEE 1394.1 bridge was available. In the terminology of IEEE 1394 and related standards, the term "network" is used for IEEE 1394 buses coupled with bridges according to IEEE 1394.1. Single buses are not called "networks" in these standards. However, the protocols used on top of IEEE 1394, like HAVi and AV/C, can only be used on single buses in their current versions. So if two buses are coupled, an intermediate system will still be needed that intercepts the control protocol and sets up the routes for streaming. Therefore, in the meaning of the term "network" that is used in the present theses, even single IEEE 1394 buses can be called networks.

It is also possible to transmit IP data on an IEEE 1394 bus. IPv4 over IEEE 1394 is specified in RFC 2734 [67], and RFC 3146 specifies the transmission of IPv6 packets over IEEE 1394 [68]. Therefore, it is possible to treat it as an ordinary IP based network and use the corresponding protocols described in the following section. However, this approach would not take advantage of the very special features of IEEE 1394 concerning real-time A/V streaming. The specifica-

tions of the series IEC 61883 are very well customised for IEEE 1394, so an IP based solution must have drawbacks (besides introducing additional protocol layers).

3.2.2 IP Based Networks

IP, the Internet protocol, is a very widely used network protocol. The main reason is that it can be used on top of almost every lower layer networking system (Ethernet, token ring, IEEE 1394, etc.) [9]. In many homes (as well as offices) IP over Ethernet can be found [69], where usually still version 4 of IP is used [70]. These networks were not designed for the purpose of multimedia transmission and, therefore, have some weaknesses with regard to real-time data transmission (see below). Nevertheless, these networks are very common. Since they are available almost everywhere and fulfil the basic requirements needed for audio and video transmission (although they are usually not guaranteed), it is very desirable to use these networks for that purpose.

Multicast is just an add-on, that is not widely supported. Within a local area network (LAN) with a shared medium, multicast works; in fact, it is a broadcast on the physical layer. If several networks are coupled with routers, these routers have to support multicast and use some multicast routing routing protocol [20]. Within one network, IGMP (Internet group management protocol) [71] is used by the routers to find out if there are members of a certain multicast group. Thus, all multicast routing protocols require the use of IGMP. Within some e. g. company networks it can be assumed that multicast is supported, but not in the Internet in general. However, on LANs multicast works similar as on an IEEE 1394 bus: On the lower network layers it is a broadcast, higher layers see a multicast. Multicast addresses are special IP addresses, where the 4 most significant bits have the pattern 1110 binary (14 decimal) in IPv4. In IPv6 the 8 most significant bits of the IP address have to be 1 followed by 4 bit flags, a 4 bit scope identifier and the possibly 112 bit long group ID. On Ethernet, these multicast IP addresses are mapped to Ethernet multicast MAC addresses by truncation, so the IP multicast is mapped to an Ethernet multicast.

Quality of Service is very limited. There are mechanisms for QoS in IP networks, but they provide only soft guarantees and furthermore are not yet widely used in home environments. RSVP (resource reservation protocol) is the standard protocol to reserve resources in IP networks [20, 72]. In fact, RSVP is only a signalling protocol for resource reservation requests and does not by itself reserve resources. Furthermore, it is used to reserve resources on the routers and end devices of a stream, but no resources on the local network are reserved. Thus, there is always the risk of collisions because of too much local traffic. RSVP is a receiver oriented protocol, the receivers are responsible to initiate reservations, which propagate to the sender. There is no strict separation between the signalling/reservation phase and the data transmission phase, usually data transmission has already started when the reservation is performed. RSVP supports heterogeneous streams, provided the intermediate systems (routers) support reshaping the stream for a particular receiver or group of receivers. The only way to guarantee a certain quality of service for a multicast group is removing all other devices from the network that could cause random traffic. IP networks only offer best effort services, and therefore, QoS is usually only achieved by oversizing the network and providing much more performance than needed in average to ensure good performance even in worst case scenarios. Besides the so-called Integrated Services approach with per stream reservations, there is also the Differentiated Services approach, which does not provide per stream reservations, but has advantages in scalability [20].

One problem of IP networks is that transmission of audio and video is not very well stan-

dardised. There is a wide variety of data and transmission formats, open and proprietary ones, which are not compatible with each other.

In IP based networks many different protocols are commonly used for real-time transmission of multimedia. The most primitive one is HTTP, the hypertext transfer protocol. There are implementations that use HTTP to stream real-time audio data (e. g. `icecast` and `shoutcast` to mention two streaming server programs, which are freely available on the Internet). But besides some advantages (it is very simple and works well with proxies and firewalls), it also has many disadvantages. It does not support multicast, only unicast. Therefore, if more than one node is listening to the stream, several connections have to be established that use the corresponding multiple of the necessary bandwidth. Furthermore, HTTP is based on TCP, which is a reliable protocol, i. e. it does perform retransmission of packets that were not transmitted correctly. This is completely inadequate for real-time transmission because the retransmitted data will often be late and, therefore, useless. In fact, the use of HTTP is just an extension of progressive download (the presentation of a multimedia file starts while downloading), but it was never designed for real streaming.

A much more suitable protocol for transmission of multimedia data is RTP (real-time transport protocol) as proposed in RFC 1889 [73], which is usually based on UDP and in principle supports multicast as well as unicast. It also includes timestamps, which are used for synchronisation purposes, but it has to deal with the fact that in IP networks there is no global time base which is synchronised automatically. RTSP (real time streaming protocol) can be used for setting up the transmission [74]. There are also proprietary protocols like MMS (Microsoft media server) or RDT (Real delivery transport) and others. They have similar features and are used by the tools of the corresponding companies (Microsoft, RealNetworks etc.). But the preferred protocols within the Internet are the open standards by the IETF. To control devices, protocols like Jini [75, 76], UPnP [77], and others are common [78].

The receiving application of an RTP stream has to use the timestamps in the sender report (SR) of RTCP (RTP control protocol) for synchronisation, but has to tackle the problem that these timestamps refer to a clock maintained by the sender to which the receiver has no direct access. Therefore, the receiver can only use the timestamps to estimate the difference between the sender's clock and its local clock and then use the local clock, compensated by this estimated difference, as reference for playback. But since the two involved clocks (at sender and receiver) usually drift—unless some synchronisation protocol like NTP (network time protocol) is used—, the buffer in the receiving device either tends towards an overrun or an underrun. The receiver can detect this drift and adapt the estimated difference. Otherwise, it has to re-synchronise by calculating a new estimated clock difference in case of buffer overruns or underruns. Unfortunately, delayed packets (e. g. due to network congestion) could also trigger such re-synchronisation, even if the drift between the clocks was small.

Unfortunately, there is no common format for audio and video data used in this context. Many different (open and proprietary) formats for audio and video exist and can be transmitted in an RTP stream. Concerning audio, the most important open formats are uncompressed audio with 16 bit, 20 bit, or 24 bit resolution [79, 80] and MPEG compressed audio streams [81]. For video, RFC 2250 specifies how to transmit MPEG video elementary streams [81]. If audio and video are to be transmitted in one stream, MPEG-2 transport streams can be used [81], but the concept of RTP is to transport each media in a separate stream rather than using multiplexes. Digital video in the DV format of IEC 61834 [34–36] can be transported in RTP according to RFC 3189 [82]. However, it should be mentioned that this is a format commonly used on IEEE 1394 [37–39], but not on IP networks. The most important proprietary formats

system frequency	overall throughput	max. synchronous	max. asynchronous	control channel
50 kHz	24.80 Mbit/s	24.00 Mbit/s	14.40 Mbit/s	800 kbit/s
48 kHz	23.81 Mbit/s	23.04 Mbit/s	13.82 Mbit/s	768 kbit/s
44.1 kHz	21.87 Mbit/s	21.17 Mbit/s	12.70 Mbit/s	705.6 kbit/s
30 kHz	14.88 Mbit/s	14.40 Mbit/s	8.64 Mbit/s	480 kbit/s

Table 3.5: Data rates on MOST for various system frequencies

are RealAudio and RealVideo.

IPv6 [83], the successor of IP version 4, is not yet widely used, but has some advantages concerning multimedia streaming [84]. Multicast has been integrated into IPv6, so every compliant implementation supports multicast. There are features (especially the flow labels should be mentioned) that make the provision of QoS for streams easier. The functionality of IGMP is integrated into ICMPv6 (Internet control message protocol). The "multicast listener discovery" (MLD) used for the signalling between routers and hosts is a subprotocol of ICMPv6 [85].

3.2.3 MOST

MOST (media oriented systems transport) is a network specifically designed for automotive environments, although its use in home environments would be possible as well [10, 11, 86]. One focus in the design of MOST were low costs of the nodes. Therefore, the design of the nodes is rather simple compared to other networks. As the name suggests it is oriented towards multimedia transportation and is specially suited for audio transport. It is seen as a competitor to IDB-1394, as both bus systems provide solutions for multimedia networking within cars. While some car manufacturers favour MOST, others favour IDB-1394. So it is not clear, whether one of these two system will establish as standard solution and supersede the other one.

Like IDB-1394, MOST uses POF (plastic optical fibre) as physical media and supports up to 64 devices. It is much more restricted with regard to topology, MOST uses a ring, whereas IDB-1394 allows almost free topology. MOST is a synchronous bus, all nodes share the same clock and data is transferred at an exactly determined rate with a very low jitter. There is one timing master on the bus that generates the clock and all other nodes slave to this clock. The sample rates used by the nodes are derived from this clock, so the devices need not buffer data for jitter or clock drift compensation, but can directly feed data from the bus into a DAC or from an ADC to the bus. This data rate can be chosen between 30 kHz and 50 kHz, usually 44.1 or 48 kHz is used. This system frequency should be the most common sample rate used in the system in order to avoid necessary sample rate conversions. Since 44.1 kHz is the most common audio sample rate, it is usually used as system frequency.

For each cycle, a 512 bit frame is sent containing 496 bit payload and 16 bit administrative data. At a system frequency of 50 kHz this leads 24.8 Mbit/s effective data rate. As shown in Table 3.5, for the more common frequency 44.1 kHz the data rate is lower. This demonstrates that in a typical system the data rate is below what MOST marketing papers tell.

MOST supports three types of data transfer:

Synchronous channel is used for continuous data streams like multimedia data. Each byte in the synchronous time slot can be dedicated for a special purpose, so on MOST an application does not just reserve a required bandwidth, but in fact, reserves the exact position of

the data within the data frames. And since these data frames are transmitted with a fixed frequency and very low jitter, obviously the QoS requirements for multimedia transmission are met. Synchronous data is not protected with a CRC (cyclic redundancy check), and due to the fixed position in the frames, no retry procedure is possible. Multicast transmission is possible.

Asynchronous channel is used for transmission of packet oriented data with large packet sizes, often required for burst data transfers. Asynchronous data is protected with CRCs, but it is only used for error detection, no retry procedure is used.

Control channel is used for low bandwidth control purposes. It provides a reliable datagram service with an ACK/NACK mechanism and an automatic retry procedure.

While the control channel has fix two bytes in every frame (leading to around 700 kbit/s for the usual system frequencies, see Table 3.5), synchronous and asynchronous data share 60 bytes in every frame. The division between synchronous and asynchronous data can be modified dynamically. The maximum amount of asynchronous data per frame is 36 bytes. Therefore, at least 24 bytes are always reserved for synchronous data. This leads to the data rates in Table 3.5.

The transmission of multimedia data as well as control mechanisms are well standardised by the MOST Cooperation. Unfortunately, only some specifications are publicly available, most are only available to members. For audio transmission, uncompressed audio is used, typically with 44.1 or 48 kHz sample rate. Video can be transmitted as MPEG-1 or MPEG-2 video. For control, a simple protocol is used, similar to AV/C on IEEE 1394. Since MOST is a synchronous bus, no timestamps are needed for jitter compensation or stream synchronisation. But each active node introduces a delay to data in the ring (2 samples). In order to compensate different delays at different sink devices, the sink devices get the information about the delay from the timing master to the source device, from the timing master to the sink device itself, and the delay of the complete network.

In comparison to the isochronous data transmission on IEEE 1394, the synchronous nature of MOST has advantages and disadvantages. On the one hand, it is not necessary to use timestamps for jitter compensation and stream synchronisation, no buffering of data is necessary. On the other hand, it reduces the flexibility of a device feeding a data stream from a real-time source not synchronised to the system clock into the bus. In this case, multimedia data has to be re-clocked, sample rate conversion might be necessary. On a bus where there is no lock between a network clock and the clock of ADCs and DACs, real-time multimedia data has just to be tagged with an appropriate timestamp, but there is no need to transmit a fix number of samples during each network cycle as it is required on MOST. This means from the viewpoint of an application sourcing real-time multimedia data on MOST bus, MOST can be treated similar to a local sound card on a PC, which provides its own clock for sampling data that cannot smoothly be adapted by the application.

As shown in Figure 3.6, for establishing a synchronous connection, a connection manager gets involved, which builds the connection on behalf of the initiator. Usually the reservation of the resources (i. e. synchronous channels) at the timing master is performed by the sink device, although the connection manager can bypass this and tell the source device directly which channel to use.

Like on most network technologies, it is also possible to transport TCP/IP data on MOST [87]. Therefore, it is in principle also possible to use the protocols usually used in IP networks

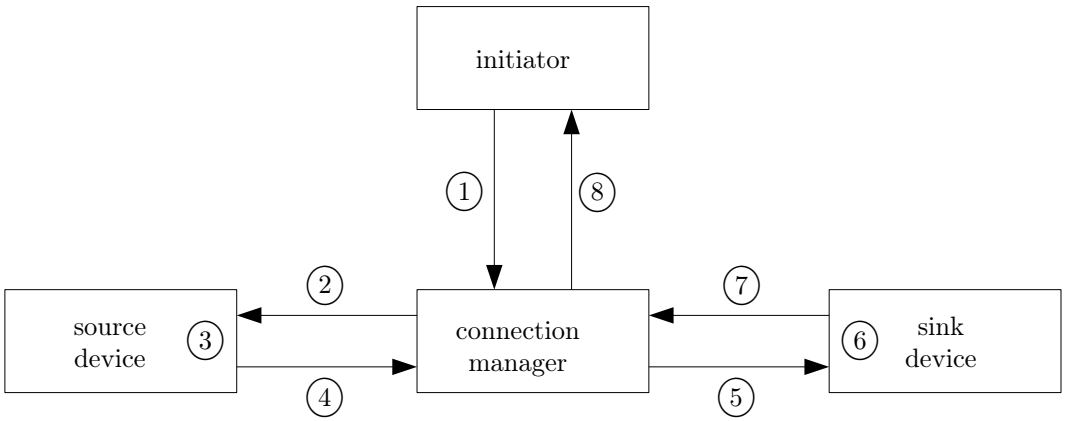

Figure 3.6: Establishing a synchronous connection on MOST [86]

for multimedia data transmission. However, this does not make much sense, since typical MOST devices cannot handle this data.

To fulfil the requirements for automotive environments, MOST uses sophisticated power saving mechanisms in order to avoid too fast discharging of the batteries. This is similar to IDB-1394.

3.2.4 ATM

Asynchronous Transfer Mode is a networking technology well suited for streaming [78, 88]. Especially with regard to quality of service it is superior to IP. The main characteristic of ATM is that it is a cell based technology. All data is transferred in blocks of a fixed size, called cells. The supported services are characterised by service classes and service categories. In contrast to IP, it supports guaranteed QoS with hard states. It has some weaknesses concerning multicast because group communication was not addressed when ATM was designed. The problem is that each sender has to maintain its own multicast tree with the recipients. There are solutions to provide multicast over ATM [20]. Due to various reasons, especially because it is an expensive technology, ATM is not used for home networking [78]. It is used in MANs (metropolitan area networks, typically spanning a town or a campus) and WANs (wide area networks, covering a large geographical area), but it is usually not found in home or automotive environments. Therefore, it is not covered in-depth here.

3.2.5 Wireless Networks

Wireless networks become more and more popular because they allow networking without the need to install cables. Table 3.6 gives an overview about some wireless network technologies. The first six protocols are used for wireless LANs (WLANs), whereas the last two ones are wireless protocols that are used for other purposes. IEEE 802.15.1 (Bluetooth) is intended as a cable replacement for connection of portable devices [89]. IEEE 802.15.4 and ZigBee (which is based on IEEE 802.15.4) are intended for wireless monitoring and control purposes like building automation [90, 91]. Because of their limited bandwidth, both protocols are not suited for real-time multimedia transmission (except for limited audio streaming), they are not used for home networking and are only listed here for the sake of completeness. Of course, this list of wireless

	IEEE 802.11	IEEE 802.11a	IEEE 802.11b	IEEE 802.11g
data rate	2 Mbit/s	54 Mbit/s	11 Mbit/s	54 Mbit/s
frequency	2.4 GHz	5 GHz	2.4 GHz	2.4 GHz
modulation	FHSS, DSSS	OFDM	DSSS	OFDM, DSSS
	HomeRF2	HiperLAN/2	IEEE 802.15.1 (Bluetooth)	IEEE 802.15.4, ZigBee
data rate	10 Mbit/s	54 Mbit/s	1 Mbit/s	250 kbit/s
frequency	2.4 GHz	5 GHz	2.4 GHz	2.4 GHz
modulation	FHSS	OFDM	FHSS	DSSS

Table 3.6: Wireless network technologies

protocols is by far not complete, there exists a wide variety of other protocols.

The data rates given Table 3.6 constitute the nominal physical layer throughputs. Because of overhead and interference, the real achievable throughput is significantly lower; as a rule of thumb it is usually about 40 % below.

All these wireless protocols operate in frequency bands where no license is required (called ISM bands—bands for industrial, scientific and medical use): either at 2.4 GHz or at 5 GHz. The 2.4 GHz band has the disadvantage that also many cordless phones and microwave ovens transmit within this spectrum. Therefore, there is much interference. All these wireless protocols use some spread spectrum technology:

- DSSS (direct sequence spread spectrum)
- FHSS (frequency hopping spread spectrum)
- OFDM (orthogonal frequency division multiplexing)

Due to regulatory restrictions, not all mentioned network protocols can be used in all countries. There might also be a restriction with regard to the allowed frequencies, so in some countries only a fraction of the specified frequencies are allowed to be used.

The different variants of IEEE 802.11 specify—in analogy to the other IEEE 802 standards—the physical and data link layers, on top of which protocols like TCP/IP can be used [92–95]. They can be regarded as a kind of "wireless Ethernet". In contrast to Ethernet, where CSMA/CD is used to detect collisions, wireless protocols have to use a collision avoidance mechanism (CSMA/CA) because collision detection is not possible. The European specification HiperLAN/2 is a bit different, it is sometimes referred to as "wireless ATM" because it provides a cell based convergence layer (for ATM cells) besides a packet based convergence layer (e. g. for IP packets) [96, 97].

Since these wireless LAN technologies are usually employed to build up Ethernet like LANs without cables and IP is used on top of them, they can be treated as IP based networks, like the wired ones. Some of these wireless protocols have integrated support for QoS, which is very valuable for real-time multimedia transmission. For that purpose, instead of CSMA (carrier sense multiple access), a TDMA (time division multiple access) mechanism is used, where the senders get fixed time slots assigned. Since there are no products available using this feature for multimedia streaming, this is not addressed here. It should be mentioned that, even if QoS features are used, there can never be real hard guarantees in wireless network because it is always possible that the available bandwidth drops suddenly due to interference.

3.2.6 USB

Universal Serial Bus (USB) [98, 99] is often mentioned as an alternative to IEEE 1394 [78]. Although it shares some application areas with IEEE 1394, it is, in contrast to IEEE 1394, not used for home networking. USB does not provide peer to peer networking, but it is a host centric bus system. There is one master, the host, which is usually a computer, and many slave devices. It is typically used to connect peripheral devices like mouse, keyboard, microphone, speakers, hard discs, etc. While IEEE 1394 can also be used for certain PC peripheral devices like scanners, hard discs, cameras, etc., it is not used for low speed devices like human interface devices such as keyboards or mice. USB 1.0 supports low speed devices with 1.5 Mbit/s and medium speed devices with 12 Mbit/s, USB 2.0 adds support for high speed devices with 480 Mbit/s. So USB 2.0 reaches the data rates supported by IEEE 1394. USB can, therefore, be used for multimedia streaming, especially for audio, but as of version 2.0 also for video. However, these streaming capabilities are used for data transfer between the host and some peripheral device, but it cannot be used for peer to peer streaming between consumer electronic devices. Consequently, it is not covered in-depth here.

3.2.7 Fieldbus Systems

Fieldbus systems are typically used to connect sensors and actuators for control and monitoring purposes. There are many different fieldbus systems, usually optimised for some application areas. Typical application areas for fieldbus systems are building automation (e. g. LonWorks or EIB), industrial automation (e. g. Profibus) and the automotive sector (e. g. CAN) [5, 6]. They usually provide some kind of QoS, which is a crucial feature, especially if they are used in critical, safety relevant processes. However, their applications do not require high data rates, and consequently, most fieldbus system provide rather low data rates, typically below 1 Mbit/s. Therefore, they are not suited for high quality multimedia real-time streaming. Nevertheless, very limited audio streaming can be performed [7]. Multimedia consumer electronic devices are not connected to fieldbus systems, and often neither personal computers are. It has already been shown that fieldbus systems can, in many scenarios, be replaced with networks like IEEE 1394 that can be used for the purposes fieldbus systems are used for, but in addition are suited for multimedia streaming [100]. Although fieldbus systems might be available in modern houses and vehicles, they are not of interest for multimedia distribution and, therefore, are not covered here. An exception are systems like the above-mentioned MOST, which are sometimes classified as fieldbus systems, but are not designed for the “classical” fieldbus applications.

3.2.8 Analogue Networks

It should, of course, not be forgotten that digital networks are not the only possibility to distribute audio and video data within home or automotive environments. For example, it is common to use analogue antenna cables to distribute TV and radio programs. This has several disadvantages compared to digital networks. On the one hand, these networks are usually dedicated networks only used for this single purpose, they cannot be reused for other purposes like data communication (file sharing, web surfing, etc.). While this can be seen as an advantage because there is no contention for these resources, like it happens on multi purpose digital networks, and they are, therefore, always available when needed, it has the major drawback that it is necessary to install an additional separate infrastructure. On the other hand, this infrastructure is usually very inflexible and requires many cables. For example

to receive receive all programs from the popular Astra satellite system, it is necessary to install four cables (vertical and horizontal polarisation, low and high frequency band) from the dish antenna to a multiswitch and one cable from the multiswitch to each device that should receive the programs. If additional programs from other satellite systems, e. g. Eutelsat, were to be received, additional cables from the antenna to the multiswitch would be necessary. The multiswitch is typically located under the roof, near the antenna. If there were two devices in the living room in the ground floor that should receive satellite TV programs independently, e. g. a TV set and a video recorder, two cables from attic to the living room would be necessary. And if a third device, for instance a second video recorder, is added, an additional cable will have to be installed, it is not possible to reuse the existing cables. Because of these deficiencies of analogue systems and the increased flexibility of digital networks, this work only concentrates on digital data transmission.

3.3 Content Protection and Digital Rights Management

Content providers, who make multimedia content available to the consumers and in some cases get paid for this service, try to limit the access to their content in order to prevent unintended use of it, in particular unlimited production of copies. In order to restrict the access to multimedia data, technologies are used that are summarised under the term "digital rights management", short DRM [101, 102]. These technologies are used to limit copies, to restrict playback to certain devices or applications, to limit the allowed time of usage, and much more. There is much controversy whether DRM is good or bad, whether it can efficiently prevent illegal use or whether it more interferes legal usage, whether it will be worked around anyway by illegitimate users and only annoy legitimate users, whether it immoderately limits the rights of the users. DRM is sometimes also called "digital restrictions management", particularly by critics. In fact, from the technical point of view, this term more precisely describes, what the technology does, it is used to restrict the access to data. One part of its features is to transmit data encrypted over networks. Content providers do not want their multimedia data to be transmitted unencrypted over digital networks because this would make it easy to create perfect copies without quality loss. In case of analogue transmission, there is a degradation in quality in each generation of copies. In case of digital transmission, this does not happen; digital copies are perfect copies of the original, and consequently, content providers do not want to allow customers to create unlimited digital copies.

There is quite a lot of different technologies available for DRM. One that has been designed particularly for multimedia streaming is "digital transmission content protection specification" (DTCP) [103]. It has been developed by five companies[6], which are collectively referred to as "5C". The initial version of this specification targeted digital multimedia transmission over IEEE 1394, but meanwhile there are also supplements available for USB, MOST, Bluetooth and IP. So all major kinds of network used for multimedia transmission within home and automotive environments are supported by DTCP.

DTCP supports different kinds of cipher, i. e. different encryption algorithms, for content encryption. The keys used for encryption are changed every 30 to 120 seconds. For authentication and key exchange, strong cryptography based on elliptic curve cryptography is used. On IEEE 1394, an extension to the AV/C protocol has been developed for this negotiation. For

[6] Hitachi, Ltd., Intel Corporation, Matsushita Electric Industrial Co., Ltd., Sony Corporation, and Toshiba Corporation

other kinds of network, this AV/C protocol extension is mapped into other control packets.

DTCP supports four different encryption Modes:

Copy-free means that the content is allowed to be copied freely. Therefore, no authentication or encryption is necessary, the stream can be transmitted unencrypted as if no content protection would be in use.

Copy-never is the exact opposite, it means that no copy of the content is allowed. Devices receiving the stream are only allowed to present it, but may not create copies.

Copy-one-generation is used, where one generation of copies can be made. Devices receiving the stream are allowed to create copies, but if these copies are transmitted again, they have to be marked as no-more-copies.

No-more-copy is used to mark streams that are already the only one generation of copy that is allowed and, therefore are not allowed to be copied anymore.

There are also two different levels of authentication, full authentication and restricted authentication, where the latter one requires less resources of the device. Copy-never streams require full authentication, whereas other kinds of stream can also be accessed by devices only supporting restricted authentication. In order to make devices and programs obey these rules, they have to be certified by the digital transmission licensing administrator (DTLA). Otherwise, they will not get the cryptographic keys required for authentication and key exchange. That way, the DTLA can ensure that only systems obeying these rules of content protection are able to access the restricted content. In order to exclude devices that are known to have been compromised, there is also a mechanism to disable such systems by revoking their cryptographic keys. For this purpose, system renewability messages (SRMs) can be distributed over several channels, e. g. inside multimedia content.

For transmission of digital TV according to DVB, conditional access (CA) mechanisms are used to encrypt the data MPEG-2 transport streams for transmission [104]. An example for another DRM system is one proposed for playback of stored content in home networks [105].

Since there is quite some controversy about these DRM mechanisms, it is not clear at the moment how successful they will be in the future. It is not known whether most future media will be protected by such mechanisms or most media will be freely available or whether both kinds of media will coexist side by side. Depending on this outcome, it will have to be decided how much support of this mechanisms is necessary in multimedia streaming gateways.

4 Convergence of Networks for Multimedia Streaming

Convergence between different kinds of network available in home and automotive environments can make additional services available to the user. It significantly enhances the usefulness of the networks and the devices connected to them because it enables access to devices of the other network and makes new interactions possible. To achieve this, gateways are necessary. This chapter points out the functional requirements of such gateway and refers to related work, which partially achieves these targets or relates to this topic.

4.1 Functional Requirements

If it was possible to use devices designed for different kinds of network together connecting sources in one network to sinks in another network, this would be a great additional benefit for the user. Therefore, the main target of this work is to develop possible architectures for gateways between different kinds of network for real-time multimedia streaming. It should be possible for sink multimedia devices to use as many sources available in their environment (where the focus here is on home or automotive environments) as possible and play the content they provide, no matter which network they are connected to. The borders for real-time multimedia streaming between different networks should be diminished.

On the networks described in the previous chapter different protocols for transport of multimedia data and for control are employed. Also different data formats are typically used. The devices on one network might support several different data formats, and it might be possible for a gateway to find a data format that is supported on two networks which are to be connected to avoid a conversion. However, there are usually typical data formats for each network that fit well to the properties (in particular the available bandwidth) of this network and are used by typical devices on this network. These issues have to be taken into account when designing such gateway. In order to make connections possible between as many devices on the various networks as possible, the gateway should not only be able to work with a very restricted class of devices on some network, which were perhaps specifically designed to work with the gateway. Instead, the gateway should be able to serve typical devices on each network, which do not require some tweaking to enable this interconnection. This strongly suggests that the gateway should support the typical data formats used on each network, not just ones that are exotic on that very network, even if this imposes the requirement for additional conversions.

Since there are several protocols involved into streaming, the design of such a gateway does not just involve the mapping of one protocol to another, but requires the interaction of a suite of protocols on one network with a whole suite of protocols on the other network. It requires the translation of a data format, a transport protocol and one or more control protocols. The focus of this work is the transport of multimedia data and the protocols directly related to this transport, the transport protocols and their associated protocols for establishing streaming connections. The mapping of the higher layer control protocols used for for control tasks that

are not directly associated to the streaming connections are not in the main focus of this work, although a complete implementation of a gateway also has to cover this part. So, for instance, this work deals with the maintenance (establishment and breaking) of an audio stream between a radio tuner in one network and an amplifier device in the other, but configuring the tuner to some frequency is not within the main focus.

Real-time streaming means that the data stream is transmitted by the source at the same speed it is intended to be played at the sink. The sink receives small chunks of data and plays them out immediately. It needs a small buffer for synchronisation purposes. This synchronisation task can be split into two parts:

Intra-media synchronisation (also called intra-stream synchronisation) cares about the temporal relationship between samples (or more general data units) within one stream. The time difference between any two samples presented at a sink device must be the same as the time between their production at the source device. Otherwise, if the time between two presented samples is too short on average, i. e. the stream is presented too fast, the buffer at the sink device will underrun. On the other hand, if this time is too long on average, i. e. the stream is played too slowly, the buffer will overrun. Both effects can lead to annoying artefacts, especially in case of audio streams, because the human ear is very sensitive to the timing of audio samples. Therefore, the sampling clocks at source and sink device have either to be synchronised or, if this is not possible, the sink device has to use some algorithm to adapt the sample rate to the local sampling clock. The most primitive adaptation algorithm is dropping or inserting single samples from time to time. Besides synchronisation of clocks, intra-media synchronisation requires also the handling of lost packets. It is not possible to just cut the samples contained in the lost packets out of the presented stream because then the temporal relationship between the remaining samples before and after the lost ones would become incorrect. Therefore, some delay or interpolated samples have to be inserted at the sink. The third part of intra-media synchronisation, which is the main cause for the requirement of buffering at the sink, is handling the transit delay variation (jitter) of the network. Since not all packets observe the same delay by the transport system, the sink has to delay presentation of fast packets to ensure that there are enough samples buffered to fill the gap when there are late packets. For this purpose, the buffer must be sized to hold at least as much data as the maximum time difference between two received packets corresponds to, and it must always be filled to a level corresponding to the maximum time until the next packet arrives.

Inter-media synchronisation (also called inter-stream synchronisation) cares about the temporal relationship between the samples (or, more general, data units) of one stream with the samples (data units) of another stream. For example, if an audio stream and a video stream comprising one TV program or movie are received and presented simultaneously, these streams have a temporal relationship, which has to be maintained. The audio stream has to be presented lip sync to the video stream. Since the packets corresponding to samples that should be presented at the same time are usually received with some lag between them, the presentation of one stream has to be delayed. Thus, the maximum lag between the streams constitutes a minimal required buffer size.

A survey on media synchronisation can be found in literature [106]. Definitions of terms and discussions about synchronisation issues in the area of multimedia are given as well as

a reference model that allows classification and comparison of different approaches. It also gives concrete numbers for the acceptable skews in various applications. For example, for lip synchronisation, a skew of ±80 ms between audio and video is usually not detected. Greater skews are detected, but might be tolerated, whereas skews beyond ±160 ms are not tolerable anymore. However, since real-time streaming is just one part of multimedia, many mechanisms described in [106] cannot be applied here. Media is always time-dependent in this case, there can be several sinks for the streams, axes-based synchronisation is used.

In real-time streaming, the sink usually does not buffer more than small fragments of the multimedia data (typically in the range of seconds or below). This "just in time" playback has the major consequence that the timing is forced by the source, which is a remarkable difference to the playback of a "local" multimedia file. During recording and playback of a multimedia stream some time references are involved because analogue signals (audio or video) are sampled at a given sample rate. The recording device uses some local clock as reference for the sample rate, and the playback device usually generates its own reference clock. Although these clocks generate the same nominal sample rate, they will always slightly differ, if they are not synchronised. Therefore, if a multimedia stream is played from a locally stored file on different playback devices, it will not be played at exactly the same speeds, but the playback time will vary a bit. In case of real-time streaming it is not possible to use a local reference clock independent from the source device, the local clock at the sink must be locked to the clock at the source device. The sample rates in the network (at source and sink devices) have to be synchronised. Since there is one source device and there might by several peer sinks, the source device usually provides the reference clock and determines the timing of the transmission. It is also possible to have some independent timing master in the network, which provides timing for both, source and sinks, but in any case, the clock at the sink device is slave of some other reference clock.

If the transmission was targeted only to one single sink device, it would also be possible that this sink forces the clock of the source device. The sink device could in this case just demand data from the source when it requires new data, which depends on the exact speed of its local sampling clock, and the source would have to adapt its clock accordingly. But this is not the case in the scenarios considered in this thesis. The concepts are targeted towards multicast transmission, each design that would limit transmission to only one sink is not acceptable.

A special case would be a source that does not sample live multimedia data, but streams pre-recorded data from a multimedia file. Here the sink could also receive the stream by reading it via some network file system and treat it like a local file. Then of course the sink would determine the timing of playback. However, this is not the kind of streaming addressed here. This work uses a definition of streaming, which has to fulfil the following requirements to make it more generally usable:

- It should be possible to use the stream at several sink devices at the same time (multicast). Even if in a practical implementation the stream is always directed to only one single node, the concept of the streaming technology should allow distributing it to more sink devices. It should not be limited to unicast by concept.

- It should not only be possible to distribute pre-recorded multimedia on demand, but it should also be possible to transmit live content. The system should not be limited to distribute files. Therefore, reading multimedia files via some network file system is not regarded as real-time streaming here.

To overcome the timing problems, all transport protocols used for streaming of multimedia data have mechanisms to synchronise clocks at source and sink devices. For this purpose, various timestamps in packet headers are involved. This can cause problems, especially in case of audio. Many audio playback devices (i. e. DACs), particularly sound hardware in PC environments, have local clocks, which can be set to some discrete sample rates, but which cannot be tuned smoothly to fit exactly the clock at the source. The consequence is that, in order to avoid buffer overruns or underruns, the data stream has to be adapted at the sink to yield a stream providing slightly more or slightly less samples than the original stream. The easiest way to do this is to insert or drop samples applying some more or less sophisticated algorithm. A discussion of algorithms to compensate this clock skew in case of audio streams, which make inaudible adjustments to the audio stream, is given in [107].

It should be noted that MOST does not need these timestamps because it is a synchronous bus and all clocks on the bus are synchronised with the timing master. The local clocks used for generation or presentation of multimedia data are derived from the system clock of the network.

The gateway should introduce as few additional artefacts into the stream as possible. Therefore, the gateway should transfer the timing information from the source network to the sink network by generating appropriate timestamps on the sink network based on the timestamps on the source network, but it should not modify sample rates, if possible. Modification of sample rates (e. g. by insertion or dropping of samples) should be performed at the sink devices using the timing information provided to them. No intermediate system should do this modification; it should be up to the sink devices to decide whether such modifications are necessary. The local clocks inside the gateway might be used for manipulation of timestamps, but not for manipulations of sample rates. This is, in particular, significant if the stream has to traverse several gateways because then such modifications might accumulate degrading the perceived quality of the multimedia presentation at the sink. It might be necessary for the gateway to modify the data format or some encoding parameters of the stream, e. g. if the sink network requires another data format or the data rate on the source network is too high for the sink network. But if in this modification, usually data reduction, the sample rate is modified, it should be based on the nominal sample rates with a conversion at a ratio of two small integer numbers (e. g. 1/2 or 2/3), so that a clean conversion is possible. The gateway should not adapt the conversion ratio of the sample rates dynamically based on some local clock because such adaptations in intermediate systems are likely to degrade the quality of the stream and, furthermore, make the conversion more complex and computational expensive. It should be kept in mind that in case of a constant sampling clock at the sink device each modification of the sample rate that expands or shrinks the signal also slightly modifies the tone pitch of an audio signal. Therefore, all sample rate conversions using not a constant conversion ratio have the potential to introduce undesirable oscillations in tone pitch. Figure 4.1 demonstrates the media synchronisation issues. Audio and video units (samples and frames) have to be played out at the sink device at the same time intervals as provided at the source device (intra-media synchronisation), and the temporal relation between different media streams should be the same at source and sink, although on the different networks the temporal relation between the units of one stream as well as the temporal relation between the units of different streams varies due to issues like network delay, jitter, and packetisation. The gateway does not add or remove media data to or from the streams, unless it really has to do some modification as described above. It has to adapt the timing information appropriately, append proper timestamps to media data on the sink network, and packetise it according to the rules of the sink network.

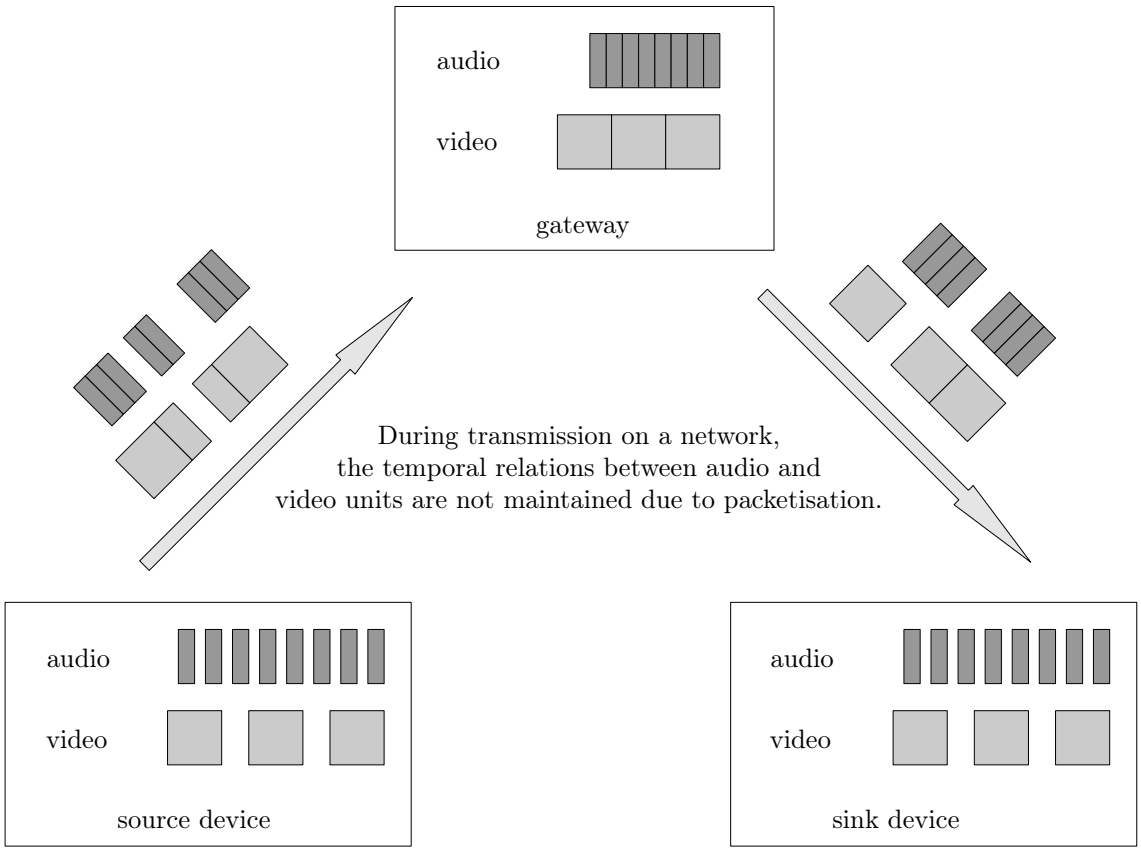

Figure 4.1: Intra-media and inter-media synchronisation issues

Gateways whose target network is MOST or a similar synchronous network are a notable exception of the above-mentioned rule. Since this network provides a global clock that is the basis for sample rates at all connected devices and on which also the transport speed on multimedia data is based, the stream must be fed into the network at exactly the speed required by the network. MOST does not allow to adapt the packet size of multimedia data, but the number of samples which have to be provided per cycle is fixed. Therefore, in the case the timing of the stream has to be adapted already in the gateway, it cannot be left to the end device. In fact, the whole bus might be seen similar as an end device in other kinds of network.

The gateway might also have to deal with content protection and digital rights management. There might be streams available that are encrypted in order to restrict the use of them. The gateway has three possibilities to handle these streams, and which to choose, depends on the particular technology used for content protection and license agreements. The gateway might pass the stream through unmodified and let only the sink devices care about this encryption. It might decrypt the stream and retransmit it unencrypted on the target network. Or it might decrypt it and re-encrypt it on the sink network using a different key, maybe also a different encryption technology. Transmitting the stream unencrypted will probably not be allowed by the content providers, who will only give access to their content protection technology if it is ensured that no unencrypted digital content is transmitted. Decrypting and re-encrypting is a viable solution, but it requires an agreement with the providers of the digital rights management technology in order to get access to this technology. Relaying an unmodified stream is obviously

the easiest method for the gateway, but it has several drawbacks. If the stream is not decrypted by the gateway, it will also be impossible to convert it to another data format or to scale the stream[1]. A prerequisite for this to work is that the sink devices are able to get the information necessary to decrypt the stream. Depending on the used technology, the sink devices might have this information integrated or can get it somewhere, e. g. via the Internet. However, it is also possible that negotiations between source and sink devices are necessary in order to exchange encryption keys, as it is done in DTCP [103]. The keys might even change during transmission and require frequent key exchange sequences between source and sink. In case of such necessary negotiations between source and sink, the gateway must, of course, also relay the according control protocols. This is obviously only possible, if this technology can be used on both networks. If a content protection technology is only supported on one network, it will not be possible or, at least, will not be useful to retransmit the unmodified stream on the sink network.

Furthermore, the design of the gateway should be as universal as possible. It should not be limited to just one special type of multimedia stream or to a special source or sink device. It should be possible to adapt it for different data formats and media types, at least conceptional, even if concrete implementations are limited. The most flexible concept would be a gateway, that is extensible by some kind of plugin mechanism.

4.2 Related Work

Since digital distribution of multimedia data more and more replaces analogue techniques, there is much work in progress bringing together sources and sinks of that data on different networks.

One issue is the transmission of DV based video over IP. It was shown how to transmit DV based video over IP enabling the connection of two DV camcorders or VCRs (with IEEE 1394 access) via the Internet. There was first a solution using UDP directly [108] and later a solution by the same authors using RTP [109]. This basically couples two IEEE 1394 buses containing these devices by the use of two gateways as shown in Figure 4.2. The main contribution of this work was that it lead to the specification of transmission of DV over RTP [82]. Therefore, transmission of DV over RTP/IP does not use a proprietary protocol, but it is standardised now, and it would, consequently, be possible for compliant devices implementing this protocol to directly process and display the streams received from IP or to provide content on IP. However, in the shown setup, only IEEE 1394 devices were used as sources and sinks of DV data, the IP network in-between was just used as a kind of tunnel. Since the available bandwidth on IP networks often is not sufficient for full DV data streams, they proposed a simple yet effective and efficient algorithm to compress the DV stream. Compression is not completely the correct word because they just drop parts of the video data, while audio data is always transmitted completely. This works quite well, since missing video frames are not as disturbing as interruptions in audio. Unfortunately, this data reduction has to be configured statically and does not dynamically adapt to the available bandwidth. A major shortcoming of this solution is that the original timing of the stream at the source is not maintained at the sink. At the sink device the sequence of full and empty packets is generated according to local timing at the sink gateway. If the source provides media data at a slightly higher rate than expected at the sink gateway, frames will be dropped at the sink, and if the source provides data at a slightly

[1] It might be possible to scale the stream, if parts that could be discarded were marked appropriately and can be identified and dropped without decrypting.

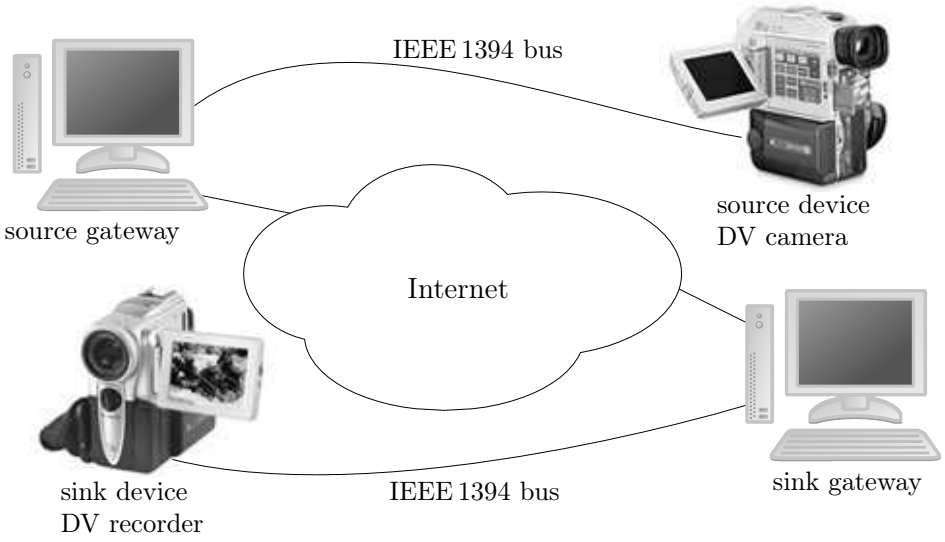

Figure 4.2: Setup for tunnelling DV data through IP used in [109]

lower rate, then additional frames will be inserted like in case of packet loss. It has already been stressed in the present thesis that the frame rate is not coupled to the IEEE 1394 cycle time by the standards in the series of IEC 61883, it is allowed that data is transmitted slightly slower or faster than the IEEE 1394 cycle time in conjunction with nominal frame rate would suggest. There is no need to force a particular timing when transmitting on IEEE 1394, it is just necessary to adapt the ratio of full and empty isochronous packets.

A working group at the Corporate Research and Development Center of Toshiba in Japan proposed two gateways, a "home gateway" and a "wireless gateway" [110–112]. Both essentially represent gateways between an IEEE 1394 based home network and an IP network. The home network contains AV/C devices like a DV camera, which should be controlled from the IP network and whose A/V data stream should also be available on the IP network. In case of the home gateway, it is connected to a public access network, so the IP network is the Internet. Control tasks are performed via a web interface using a web browser with Java Virtual Machine to execute Java applets. On the gateway there are Java servlets and backend servers.

The second gateway, the wireless gateway, is used to connect devices on the wired home network, an IEEE 1394 bus, with wireless devices that are also part of the home environment. This wireless home network offers significantly more bandwidth than the public access network in case of the home gateway, which is the most important difference from the streaming point of view. In their work, they propose using AV/C as control protocol on IEEE 1394, but also on the wireless network. Therefore, of course, it is necessary to adapt the AV/C protocol for the wireless IP network. The gateway terminates the AV/C protocol stack on both sides and provides proxy subunits that are part of the gateway unit for all subunits on the other side. Their work describes relaying AV/C commands targeted to a subunit of the gateway, which is a subunit proxy, to the corresponding "real" subunit on the other side. However, it should be noted that forwarding commands addressed to the AV/C unit is not that trivial, especially if there was more than one AV/C device in the network. In the simplified network scenario in the mentioned work, there is only one AV/C device in each network, which might comprise several subunits. Since all subunit proxies are part of one unit, devices on one network see all

subunits of the other network as part of one unit. If a device on one network tries to establish a connection between two devices on the other side, it will look like an internal connection between two subunits of the same unit, and therefore, an AV/C `Connect` command will be issued to connect a source plug of one subunit with a destination plug of another subunit. The gateway cannot just simply forward this command to some unit on the other side, there is not only one responsible unit. Instead, it has to establish two connections within two different units, one connection between a source plug and a serial bus plug and one connection between a serial bus plug and a destination plug. Furthermore, an external isochronous connection according to IEC 61883-1 has to be established. Although the implementation of AV/C proxies seems not to be too complicated as outlined in the mentioned papers, there are some quite nontrivial details, which have to be observed and which were not explicitly addressed.

These works focus on the control task of the gateways, but they also cover the streaming task. Both gateways receive DV streams over IEEE 1394 and transcode them before transmitting them on the IP network in order to reduce the required bandwidth. The home gateway transcodes the DV data stream in software to MPEG-4 at 32–512 kbit/s and sends it over RTP. The wireless gateway uses an MPEG-2 hardware encoder board together with a hardware media converter that decodes the DV data stream to generate a variable bit rate MPEG-2 data stream at on average 2–3 Mbit/s. This MPEG-2 stream is then transported in RTP packets. Unfortunately, the work about these gateways does not tell many details about the streaming part of the gateways. It is not sure that they take into consideration all the sophisticated issues, especially concerning timing, which are the main focus of the present thesis. It is not clear whether timing of the stream on the sink network is forced by the gateway device independently of the source stream or if the original timing of the source stream is maintained. As already stressed in this thesis, a proper streaming gateway should maintain the original timing of the stream as long as possible during its way over the network. Otherwise, glitches like lost frames or short interruptions might occur.

There is literature about streaming solutions targeting applications like video on demand, e. g. [113]. In such scenario, there is usually one source of video services and one sink, which presents the stream. The start of streaming at some playback position within the media is initiated by the sink. Therefore, the timing of playback and of data transmission can be controlled by the sink, and in the cited example it is controlled by the sink. A flow control mechanism, which is based on the usage of a ring buffer in the sink device, is employed. This is the most important difference to the architecture described in the present work, where timing and flow control cannot be enforced by the sink since it should allow multicast streaming to many sink nodes. Furthermore, the present work does not focus on streaming of content stored locally at some access point, but on the forwarding of multimedia streams. It does not target an end point of a stream, but an intermediate point of the stream, where it travels from one network into another one.

At the 1394 Trade Association, there is a wireless working group active on the topic of convergence between IEEE 1394 and wireless technologies. The original aim of this working group was coupling IEEE 1394 buses by establishing an IEEE 1394.1 [66] compliant distributed bridge. This distributed bridge should essentially tunnel the IEEE 1394 protocol over a link compliant to IEEE 802.11 or some variants, in particular IEEE 802.11a/e [114]. There was a draft[2] for a specification of a protocol adaptation layer in 2002 and also a prototype by Philips. These first activities did neither lead to some international standard nor to commercially available

[2] Unfortunately this draft is not publicly available.

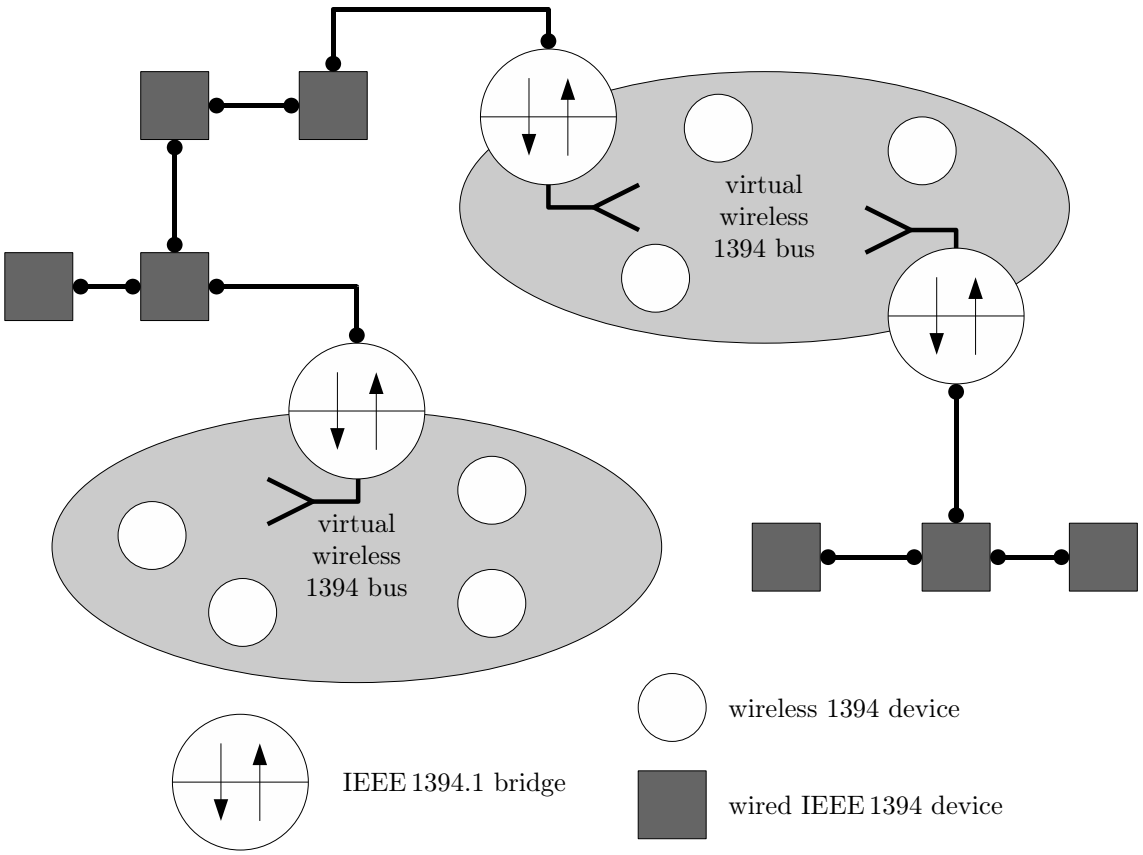

Figure 4.3: Heterogeneous network of wireless and wired IEEE 1394 buses

products. However, a few years later, this working group released a specification for a protocol adaptation layer for IEEE 1394 over IEEE 802.15.3 [115]. So a different lower layer network technology is used, but in principle the architecture is still similar to the original approach. IEEE 802.15.3 constitutes a wireless personal area network, so the spatial range is rather short, typically it is used for connection of devices within one room, it works in the 2.4 GHz range and supports data rates up to 55 Mbit/s [116]. The protocol adaptation layer (PAL) allows the creation of wireless IEEE 1394 nodes, so it substitutes the IEEE 1394 link layer allowing wireless devices to form a virtual wireless IEEE 1394 bus. The main value, however, of wireless IEEE 1394 is networking between wireless and wired IEEE 1394 devices via IEEE 1394.1 bridges. Figure 4.3 demonstrates how wireless IEEE 1394 buses can be coupled via a wired IEEE 1394 bus using these bridges and how wired IEEE 1394 buses could be coupled using two bridges connected to the same wireless bus.

While these activities targeting wireless IEEE 1394 over IEEE 802.11 where mainly carried out in the USA and Japan, there were projects in Europe on wireless IEEE 1394 over HiperLAN/2. These efforts lead to the first specifications for connectivity between wireless and wired IEEE 1394 devices [117, 118]. The various wireless IEEE 1394 technologies can, of course, be used to couple IEEE 1394 buses and to extend the availability of multimedia content of one bus to others. However, on layers higher than the link layer it is always the same IEEE 1394 based protocol suite that is used. Therefore, this approach would not lead to convergence

between different kinds of networks, which is the focus of the present work.

Transmission of IPv4 and IPv6 over IEEE 1394 has been specified [67, 68]. Hence, it would be possible to make all services that exist in the Internet available to IEEE 1394 by simply using an IP router, which connects the IEEE 1394 bus to the Internet. However, this approach does not take advantage of the special features of IEEE 1394. It would not use the isochronous transmission mode and the resource reservation mechanism. The series of IEC 61883 would not be needed. But typical consumer electronic devices on IEEE 1394 can only work with the protocols on top of IEC 61883, they usually have no support for IP. And even if they had support for IP, they would probably not support many of the protocols typically used for multimedia streaming in IP networks, while they perfectly work with the typical protocols and data formats used on IEEE 1394 buses. Therefore, most typical IEEE 1394 consumer electronic devices would be excluded from multimedia streams over IP encapsulated in IEEE 1394 packets.

A design concept for a residential access point with one focus at interconnecting multimedia and networking middlewares like HAVi and Jini is given in [54]. A main topic of this work is bridging between the protocols Jini and HAVi (see Subsection 3.2.1 and Subsection 3.2.2), which are used for control purposes in multimedia home networks, but also AV/C devices are included. The scenario there is primarily based on a residential IEEE 1394 bus using HAVi. The residential access point providing a bridge between HAVi and Jini allows nodes of an IP based network, which might be located outside the residential network, to remote control the HAVi network using Jini. So it is possible to establish an isochronous connection on the IEEE 1394 bus between a HAVi tuner and a HAVi display device and to control these devices (select a TV program, adjust the brightness, etc.) from outside via Jini. But it does not allow streaming connections across these networks, the gateway does not handle or convert real-time multimedia streams. Therefore, it is complementary to the present works, which focuses on the streaming part of a gateway, but not on the control part. The target of this present thesis is exactly one main missing part of [54], a gateway functionality for multimedia streaming between different kinds of network.

Coupling of networks within the home is also addressed in [119]. This is a work on convergence between three kinds of network. On the one hand, there are home automation networks, typically based on fieldbus systems like LonWorks or EIB. On the other hand, there are multimedia home networks for consumer electronic devices, for example based on IEEE 1394 and HAVi or AV/C. The third class are small office and home office networks, typically based on IP networks, where protocols like UPnP or Jini are used. In this survey, different coupling methods are analysed and a classification of gateways is developed. It does not deal with multimedia streaming and suggests that multimedia streaming might be seen as a separate part of the coupling model. The gateways described are used for control tasks, and the focus of the work is convergence between consumer electronics and home automation. Therefore, it does not overlap with the present thesis, but it can be seen as complementary work.

In [100] an in-depth analysis of timing issues on IEEE 1394 buses is given. Furthermore, concepts for using IEEE 1394 as backbone for home and industrial automation are described. It demonstrates how field area networks can be coupled via an IEEE 1394 bus. Therefore, multimedia streaming is outside the main focus of this work.

5 Concepts for Convergence

It has already been outlined in this work and also in literature [54] that if there are several networks with streaming sources or sinks, it would be a benefit to the user to couple them. The effort necessary to perform such coupling depends on the similarity of the involved networks. If the features, especially the available bandwidth and quality of service, the used data formats and protocols, are similar or equal, the coupling will generally be easier. On the other hand, if the involved networks are very different, coupling will require more effort. It can usually be achieved easily, if two networks of the same type are to be coupled.

IP networks can be coupled by using routers, operating on top of the IP protocol, layer 3 (network layer) in the OSI reference model [17, 18]. If the lower layers (data link layer or physical layer in the OSI reference model) are compatible, even bridges (operating on top of layer 2, data link layer), switches (special cases of bridges), or repeaters (operating on top of layer 1, physical layer) can be used, but that would, in fact, lead to one big network without the necessity for a facility relaying multimedia streams and the related protocols. IEEE 1394 buses can be coupled by the use of bridges according to IEEE 1394.1 [66]. These bridges are more powerful than bridges in the IP world, they perform address translation (between the physical IDs used on the local bus and global node IDs needed to address nodes on remote buses), synchronisation of clocks, etc. IEEE 1394.1 bridges are rather young technology, the standard has been approved in December 2004. Therefore, they are not available as products on the market yet.

In these cases where the same kinds of network are coupled, usually no transformation of multimedia data is required, essentially the same formats are used as in stand-alone networks. The packets containing the multimedia data are routed from the source to the sink network. In case of IP, the setup of the routes is done automatically by multicast enabled routers. The routers use IGMP [71] to find out which multicast groups have members on the connected networks and some multicast routing protocol to set up the routes between the routers [20]. Unfortunately, things are not always that easy. For example, if two IP networks are to be coupled that use different lower layer technology and, therefore, offer different QoS, then at least scaling of the media stream might be necessary in order to reduce the required bandwidth. Therefore, intermediate systems, essentially multicast routers, would have to actively process the multimedia data [20]. Although the protocols for signalling such scaling requirement exist, in particular RSVP [20, 72], there are two major problems. On the one hand, not all data formats support scaling of the stream, and in some cases it might be necessary to transcode the stream to another data format. For example, if bandwidth was very limited, it would make sense to transcode an MPEG-2 stream to MPEG-4, which offers much better quality at low bandwidths. The consequence is that these adaptations, which are not transparent to the end devices, require that signalling has to be adapted, so an intermediate system has to intercept e. g. RTSP commands because the client has to negotiate the data format with the intermediate system, not with the server, and the server also has to negotiate with the intermediate system. It is even possible that the data format used on the source network is not understood by the sink device or the format used on the sink network is not supported by the source device.

Therefore, the intermediate system is the end point for signalling protocols on both networks and practically separates the networks. On the other hand, even if the multimedia stream can be scaled transparently for the end devices, that scaling is usually too much computational effort for simple multicast routers. Therefore, special devices supporting scaling features for some dedicated data formats would be needed. This example shows that even coupling two IP networks for multimedia streaming might require special gateways in order to achieve proper streaming on both networks, simple routers or bridges are often not sufficient.

In case of IEEE 1394.1 bridges, things are a bit more complicated. The protocols to control devices (like AV/C and HAVi) as well as the protocol to set up streaming connections, the connection management procedures of IEC 61883-1, were specified for single buses not coupled via bridges. While AV/C and HAVi will almost certainly get extended to support bridged IEEE 1394 networks, the connection management procedures of IEC 61883 will always be used only to set up connections on one single bus. IEEE 1394.1 specifies a special protocol for stream connection management, which is used to set up the routes for isochronous streams through the network. On the source and on the sink bus, IEC 61883-1 will be used by the bridges to establish the connections from the source node (talker) to the initial entry bridge portal and from the terminal exit portal to the sink node (listener).

So while IP networks can under certain conditions—in particular no requirement for scaling of multimedia streams—be coupled almost transparently using routers with exactly the same protocols used on the separate networks, the use of IEEE 1394.1 bridges will require extensions to the existing protocols. These connections will not work out of the box.

The general problem is the coupling of different networks where different protocols are used for real-time multimedia data transport and for control, and this is the main focus of the present work. This problem can be split into two subproblems:

1. The multimedia data stream must be transferred from one network to the other one.

2. The control protocol, which is used to set up and break streaming connections and which is usually independent from the transport protocol, must be adapted.

This leads to a gateway design as shown in Figure 5.1, which has been developed as part of the present dissertation. Both tasks can be split into several subtasks. They will be examined in the following sections. The gateway consists of two main layers, where both layers consist of several sublayers. The upper layer deals with the control part, which involves everything necessary to set up and maintain a streaming connection and to control generation and consumption of the data. So besides management of the data stream itself, starting transmission and reception, and ensuring proper routing through intermediate systems, it influences the generation of the stream, i. e. which TV program is tuned, which track of an audio CD is played, and the consumption of the stream, e. g. the brightness or volume of presentation. Protocols involved in this layer are HAVi, AV/C, Jini, RTSP, UPnP, the connection management procedures of IEC 61883-1 and many more. A possible layer structure of the protocol stack is shown in Figure 5.2. This figure is a bit simplified. For example, both HAVi and Jini consist of several submodules, some are stacked in layers, some could be seen parallel. And not all parts of Jini are based on RTSP. Furthermore, the different heights in this figure show that some layers are more powerful than others. However, the relations between the heights of different layers do not exactly represent the relations between the functionalities of the layers. And layers depicted at the same vertical position do not necessarily offer equivalent functionality. It has to be noted that this structure does not directly relate to the OSI reference model. It gives an overview

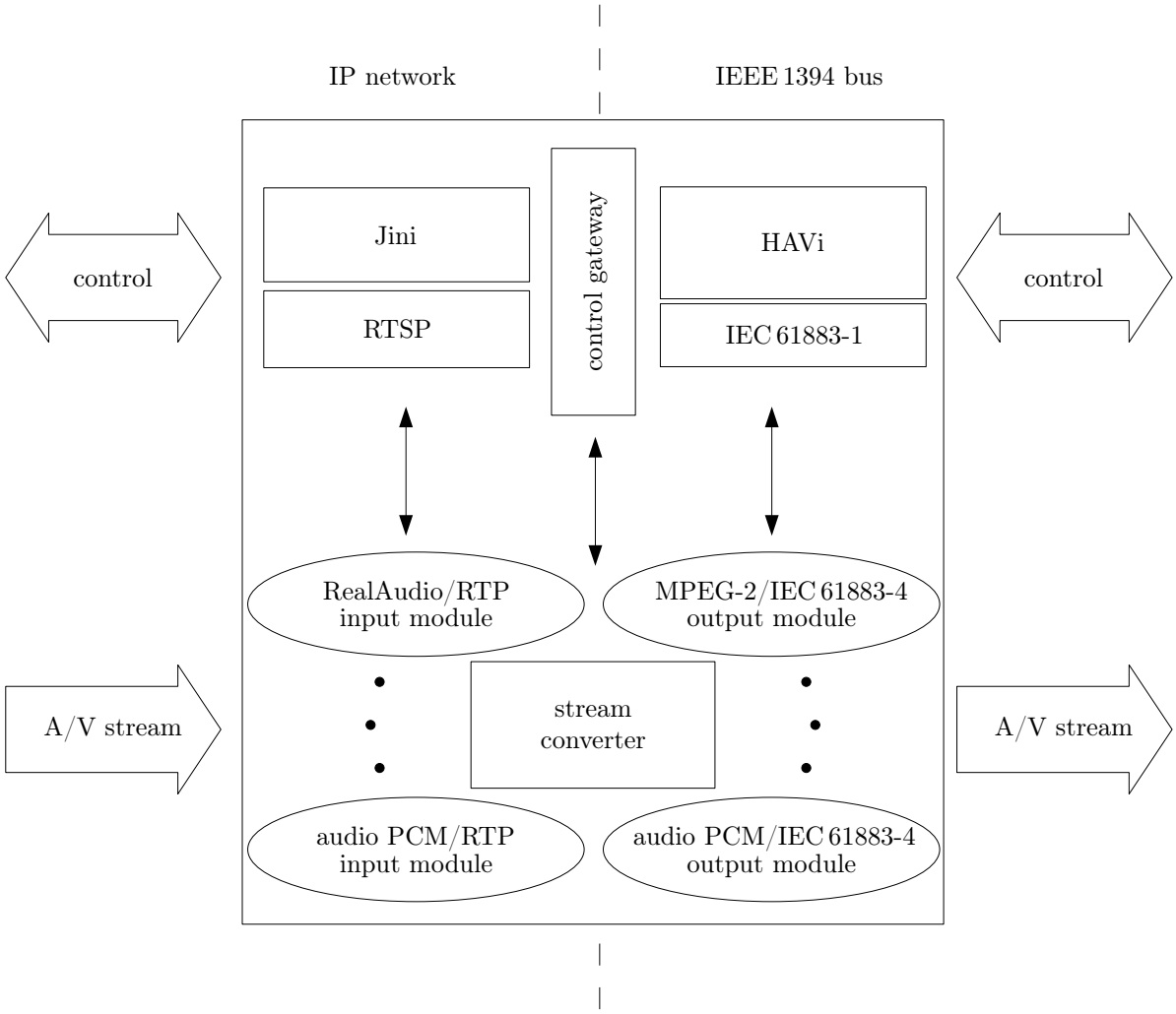

Figure 5.1: Design of a gateway between a Jini based IP network and a HAVi based IEEE 1394 bus

over the involved protocols, but it does not assign each protocol to some dedicated OSI layer because such assignment is a bit ambiguous and a matter of interpretation.

The lower layer of the gateway involves the transport of multimedia data. This is the main focus of the present thesis. It is proposed that an extensible infrastructure consisting of input and output modules is used. Between both layers, the control layer and the transport layer, interaction is needed. The control layer has to set up and control the transport layer, and the transport layer might provide status information to the control layer. A possible layered structure of the protocols used for the streaming part is given in Figure 5.3. As already mentioned for Figure 5.2, this is slightly simplified and only gives an overview over the involved protocols. It is not related directly to the OSI reference model.

This concept enables to design a gateway in a very universal way. It is possible to build an extensible infrastructure consisting of input modules (containing decoders), which deliver uncompressed data (samples or frames) tagged with timestamps (based on a reference clock within the gateway), and output modules (containing encoders), which encode, packetise, and

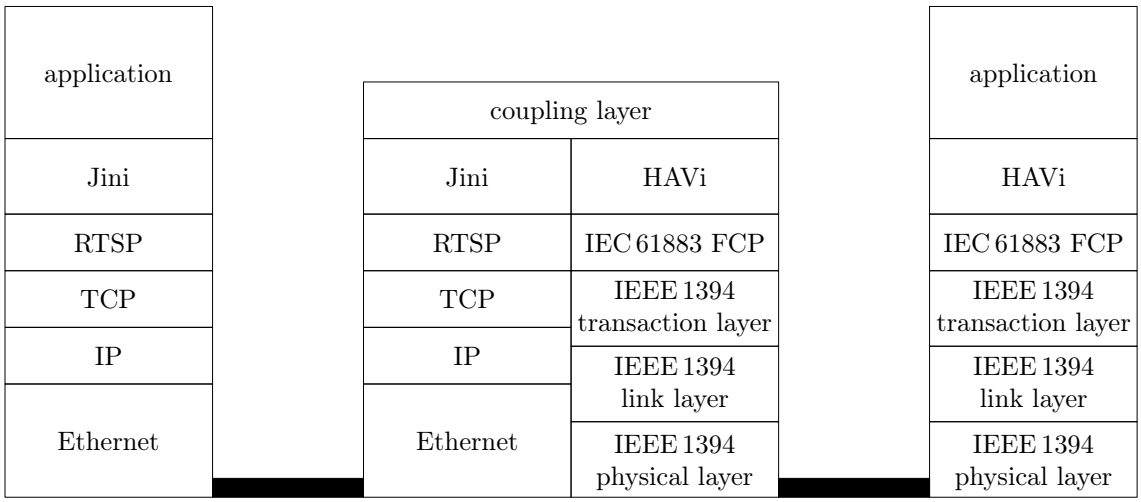

Figure 5.2: Layer structure of the protocol stack for the control part of a gateway

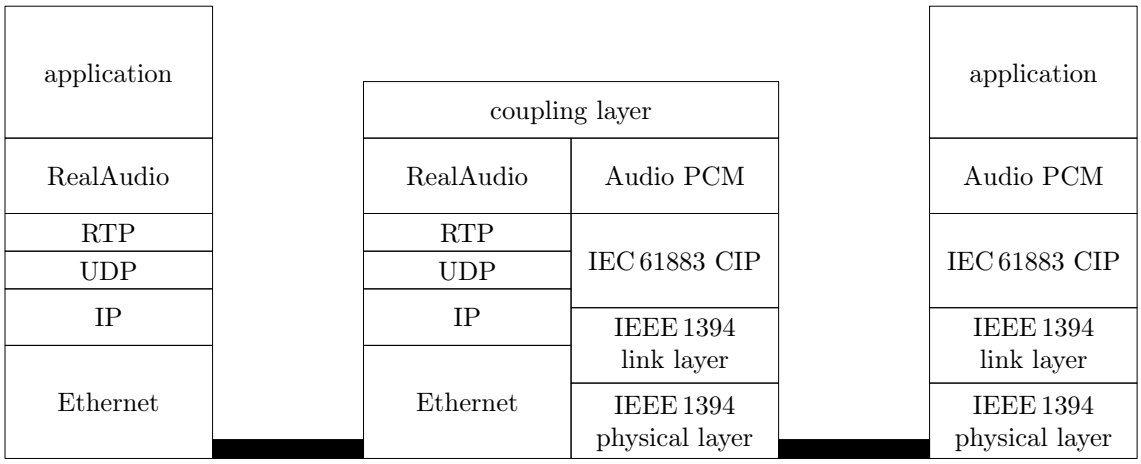

Figure 5.3: Layer structure of the protocol stack for the streaming part of a gateway

transmit the data on the target network according to these timestamps. The reference clock in the gateway could be based on a clock used on either source or sink network; in this case one of the time translations (either in the input or in the output module) can be omitted. But it is also possible to use an independent reference clock with no direct coupling to either reference clock on both networks.

5.1 Transport of Multimedia Data

The main task of a gateway for multimedia data between two networks is the transformation of multimedia data in real-time from one network to the other one. This is also the most critical task with respect to timing issues and computational effort. As already discussed in Chapter 3, the format of the packets transferred over the network usually consists of two layers. There is a transport protocol like the common isochronous packet CIP according to IEC 61883-1 on IEEE 1394 [58] or protocols like RTP, RDT, or MMS on IP networks [120]. And within these

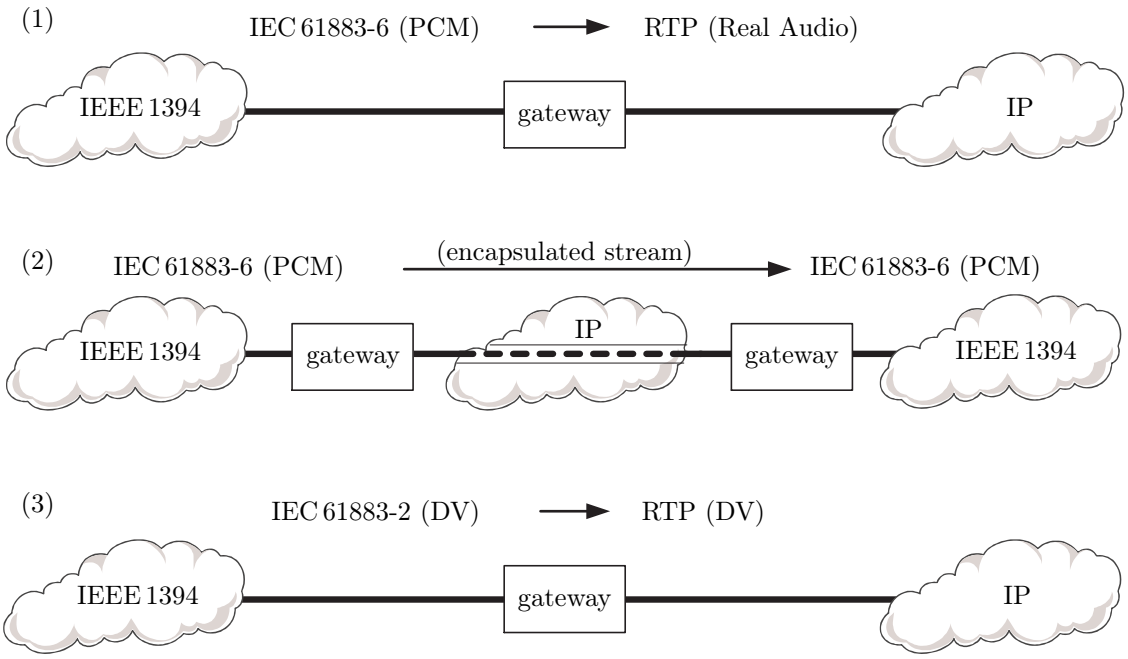

Figure 5.4: Concepts for gateways between IEEE 1394 and IP based networks

transport protocol packets, media in some data format like audio PCM samples, MPEG audio, or RealAudio is packetised. It is also possible that there are intermediate "container formats" like MPEG-2 transport streams, which can contain various data formats. It is not easy to classify all the involved protocols with regard to the OSI model—for example, RTP can be seen as a transport layer protocol (layer 4) or as an application layer protocol (layer 7) [9]. However, the transport protocols can be seen at least at layer 4 (transport layer) of the OSI model. The protocols on the layers up to layer 3 (network layer) are usually different between the networks that are to be coupled. Consequently, routers or bridges cannot be used to couple these networks, gateway is the appropriate term for the coupling unit [9].

As part of the present thesis, a classification of different architectures for the streaming part has been worked out. There are three major approaches to transfer the multimedia data stream between two networks, which are shown in Figure 5.4. The difference between these approaches is that they operate on different layers of the data stream. They can work on the layer of the used transport protocol like the common isochronous packet according to [58] on IEEE 1394 or RTP on IP networks. But they can also transform the data format like audio PCM samples, MPEG audio, or RealAudio. Or the gateway can work on both layers. All three types of gateways will be described in the following subsections. The first type of gateway in Figure 5.4, which transforms the data format and adapts the transport protocol, will be called "data format layer gateway". The second type, which capsules the complete transport packets will be called "encapsulating gateway". And the third one, which only converts the transport protocol, will be called "transport layer gateway".

In principle, it would also be possible to think of a fourth kind of gateway. It would adapt the data format, but leave the transport protocol unchanged. As the "transport layer gateway", this could be interpreted as a mixture between the "data format layer gateway" and the "encapsulating gateway" approach, but this combination does not make much sense. In order to

adapt the data format (i. e. to transcode), it is necessary to depacketise the stream at transport protocol layer. And afterwards it must be re-packetised using some transport protocol. There is no reason to use another transport protocol than the one which us usual on the destination network.

5.1.1 Data Format Layer Gateway

One approach is to set up a gateway that transforms the data in a way that on both buses multimedia data is carried in a format that is commonly used on that bus. This means that, for instance, on an IEEE 1394 bus data is transmitted according to the series IEC 61883, whereas on IP data should be transported using RTP. The data format within those packets has to be transformed as well: While on IP audio is often compressed using RealAudio or some other (mostly proprietary) format, on IEEE 1394 it is usually transmitted as uncompressed PCM samples. The main advantage of this approach is, that conventional devices can be used as sources or sinks without the need for special adaptations.

This is the most general approach and has several advantages over the other ones. It has already been worked out as functional requirement for the gateway design (Section 4.1) that the gateway should use the transport protocols and data formats on each network that are common on this particular network. Since multimedia data appears on all networks in a format that is commonly used on that particular network, typical devices will be able to take advantage of the stream and present it to the user, store it on some media, etc. Furthermore, since typical data formats are used on each network, they will usually fit better to the features (like available bandwidth and other QoS parameters) of these networks than data formats typically used on the other networks.

The drawback is that this solution is the most complex one. The stream needs to be depacketised, the transport protocol must be interpreted. Data usually has to be transcoded to change the data format, and it has again to be packetised using the transport protocol in use on the destination network. Especially the transcoding might require considerable computational effort. In the worst case, that could consist of completely decoding and encoding it again. If both data formats are sufficiently similar, complete decoding could be avoided and some more efficient transcoding algorithm might be utilised [121, 122].

Since this kind of gateway fits best the functional requirements and allows the most universal use of multimedia streams, it is clearly the one that is preferred by the present thesis. The weaknesses of the other kinds of gateways will be explained more in detail in the next two subsections. The concept for the gateway shown in Figure 5.1 perfectly matches with this kind of gateway. The stream converter has several input modules. The input modules have to interpret the transport protocol, retrieve the media data and decode it. Decoded media data, usually plain audio PCM data or decoded bitmaps, are tagged with timestamps that were provided by the transport protocol. The input modules have to transform these presentation timestamps from the time base used by the source transport protocol to the common time base used by the stream converter. Although the common time base can be some arbitrary time base available in the gateway, it makes sense to use a clock that is used as time base for some input or output modules because in this case one conversion can be avoided. For example if there are input or output modules for streams according to IEC 61883 on IEEE 1394, it makes sense to use the IEEE 1394 cycle time as common reference time base within the gateway because it is a very accurate clock and timestamps on IEEE 1394 do not require a conversion in this case.

Different transport protocols use different intervals for tagging multimedia data with time-

stamps. Therefore, not each sample or frame is tagged with such timestamps. Since the transport protocols on source and sink network might use different conventions for tagging data with timestamps, some of the timestamps will be discarded, while others have to be interpolated. Discarding timestamps will, of course, be done by the output module because this is the only module knowing which timestamps are needed and which are not and because it does not make sense to throw away available information before it is known whether it will be needed. The interpolation of additionally required timestamps could in principle be done at three locations. The input module could provide timestamps for each data unit (sample, frame, etc.). If it did not get them from the transport protocol, it would have to interpolate them. The same could be done by the central stream converter module, it could interpolate a timestamp for each unit it gets from the input module without timestamp. In both scenarios the output module would get a timestamp for each unit and could just throw away the ones it does not need. The third solution is that the output module itself generates the missing timestamps. All three approaches have advantages and disadvantages. The advantage of the last solution, where the output module generates the timestamps, is that no timestamps have to be generated that are to be discarded afterwards. Therefore, unnecessary computations are avoided. The advantage of a central module calculating the timestamps is that there is only one such module necessary in the system and the, perhaps very many, input and output modules can be kept rather simple. The real advantage of the first solution, input modules generating the timestamps, becomes obvious when discontinuities in the data stream are taken into account. If some units are lost, e. g. due to packet loss some media samples or frames are lost, the rate of these units at the central stream converter module and, of course, also at the output module would no more be constant. However, these modules can only interpolate timestamps correctly if there is a constant flow of media data at a constant rate; otherwise, some generated timestamps would be generated incorrectly. This problem could be alleviated a bit if the input module was required to interpolate missing samples themselves. It could repeat the last valid sample or employ some other interpolation algorithm to fill in the missing media entities and conceal these errors. However, this interpolation of media data would significantly increase the complexity of the input modules. Furthermore, it does not help in all situations. If the rate of media units simply is not constant, even without data loss, or if this rate changes dynamically, it does not make sense to interpolate media data. Since the input module is the module that knows the exact timing of incoming media data and since it can usually detect lost data by interpretation of information provided by the transport protocol, it is proposed that the input module should tag each media unit with a timestamp and, therefore, interpolate timestamps if necessary. Interpolation of media samples for error concealment should not be done in the input module. Preferably, it should be done in the sink device; if this was not possible, the output module should perform this task. Doing it inside the gateway is a modification of the media data stream, which should be avoided according to the requirements given in Section 4.1, and it would cause unnecessary traffic on the sink network because this interpolated data is redundant and could be created from the surrounding data at the sink device as well.

The stream converter hands over the decoded multimedia data to the output module that has to encode it according to the data format on the sink network. After encoding, it has to packetise it according to the rules of the transport protocol used on the sink network. Here it uses the timestamps provided by the input module. It has to convert the time from the common time base used by the stream converter to the time base on the sink network. Unnecessary timestamps are dropped.

Since in some situations it is possible to transcode media more directly without completely

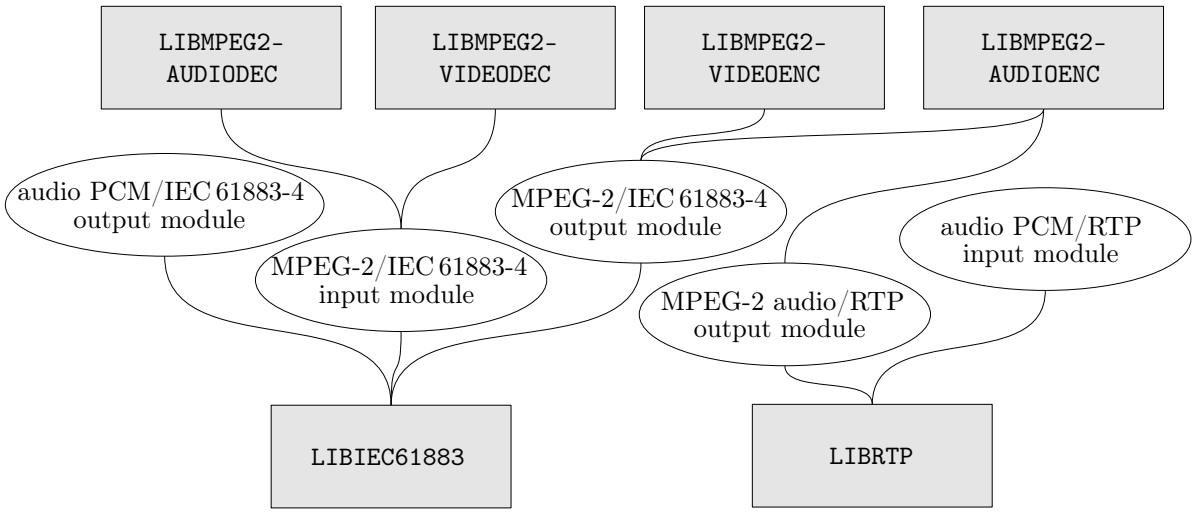

Figure 5.5: Possible architecture of input and output modules for data format layer gateway

decoding it to raw audio or video data and re-encoding it [121, 122], it makes sense that the input module not always completely decodes it. So also other data formats might be supported as intermediate format in the stream manager. However, since decoded audio samples and bitmaps of video frames are the most universal formats, each input and output module should support them. Input and output modules might support several data formats, at least completely decoded data and data not decoded at all, in the original data format as it is received from the source network or sent at the sink network, respectively, but perhaps also some other intermediate formats. After selecting appropriate input and output modules for a given conversion, the stream converter should choose the best format supported by both modules according to some rules.

Input and output modules have to handle the transport layer protocol and the data format. Both tasks—transport layer depacketising or packetising and data format decoding or encoding—can be performed in software, but also in hardware or in a combination of both. Examples for hardware support for this tasks can be found in the InHoMNet project [1]. In this project, IEEE 1394 evaluation boards by Philips/Vitana[1] were used for packetising and depacketising MPEG-2 transport streams according to IEC 61883-4. The MPEG-2 decoder chips on DVB-S PCI cards were used for decoding MPEG-2 transport streams. Examples for pure software solutions for these tasks are given later in the present work. Usually there will be several input or output modules using the same transport layer protocol, but different data formats. And there will also be modules using the same data format over different transport layer protocols. In order to reduce implementation effort, the modules themselves should be implemented in a modular way. Both submodules, the transport layer submodule and the decoding/encoding submodule, should be designed in a way to make them reusable, regardless whether they are implemented in hardware or software. That way, various input and output modules can be created simply be combining appropriate transport layer and decoding or encoding submodules. Figure 5.5 shows a possible architecture where encoding and decoding functionality as well as transport layer handling are available as libraries and input and output modules are implemented on top of these libraries. Here it is assumed that there

[1] Philips Semiconductors PDI1994L11/11 Firewire Reference Design Kit (Rev. 1.0)

is a library called `LIBIEC61883` that provides packetisation and depacketisation functionality according to IEC 61883 and can transmit and receive isochronous packets on IEEE 1394. All input and output modules for IEEE 1394 make use of the functionality provided by this library. Similarly, there is a library called `LIBRTP` that provides the capability to handle the RTP packet format and transmission and reception on IP based networks. All input and output modules for IP based networks can make use of this library. Moreover, there are libraries that provide decoding (`LIBMPEG2-AUDIODEC` and `LIBMPEG2-VIDEODEC`) and encoding (`LIBMPEG2-AUDIOENC` and `LIBMPEG2-VIDEOENC`) functionality for certain data formats. All input and output modules should use the appropriate libraries for data formats. It is also possible that some modules need more than one data format library. In case of multiplex streams like MPEG-2 transport streams transmitted on IEEE 1394 according to IEC 61883-4, both MPEG-2 audio and MPEG-2 video streams can be included. Consequently, the respective input and output modules are based on both data format libraries. Implementation of functionality that is used by several modules as libraries not only reduces the implementation effort, it also scales down the overall memory usage because the library is only loaded once into system memory.

5.1.2 Encapsulating Gateway

Transport protocols for multimedia data like RTP or IEC 61883 are usually bound to certain types of network. Therefore, a multimedia stream that leaves one kind of network and is put on another kind of network requires a transformation of the transport protocol. A way to avoid this transformation is to capsule the packets received on one network and retransmit those capsules on the other network. This could mean putting isochronous IEEE 1394 packets into IP packets (RTP packets or plain UDP packets). On the other hand, RTP packets received from the IP network could be put into IEEE 1394 packets. Asynchronous stream packets would be the most suitable type for this purpose. These packets are an extension to IEEE 1394-1995 that was added by IEEE 1394a-2000. They have the same packet format as isochronous packets, but they are sent during the asynchronous, not during the isochronous period. They are intended for streams, but do not provide a reserved and guaranteed bandwidth. It is required to reserve isochronous channels for asynchronous streams, but no isochronous bandwidth because this would be equivalent to reservation of a time slot during the isochronous period, which is not occupied by asynchronous streams. They are subject to the same arbitration requirements as all asynchronous packets, and therefore, they are a kind of best effort service and, hence, are very similar to the streams on IP networks.

This approach is the easiest solution to couple the networks, but has the major drawback that most ordinary devices cannot handle them. For instance, IEEE 1394 enabled amplifiers can only handle isochronous packets containing data according to IEC 61883-6 [62], but no RTP packets encapsulated in IEEE 1394 packets. So this solution would exclude many devices—especially consumer electronic devices, which do not offer the possibility to upgrade them with special software ("plugins") to handle this kind of encapsulated stream—from usage of the provided multimedia streams.

There is one scenario that such mechanism makes sense in: If there is a second gateway that transforms the data back and transmits it again on a network of the same type as the source network, the intermediate network will just be used as a transit network. Devices on the intermediate network would not make use of the data, but on the sink network conventional device can use the data. This mechanism could be referred to as "tunnelling". Figure 5.6 demonstrates the traversal of a packet through such gateways. This figure is a bit simplified, it

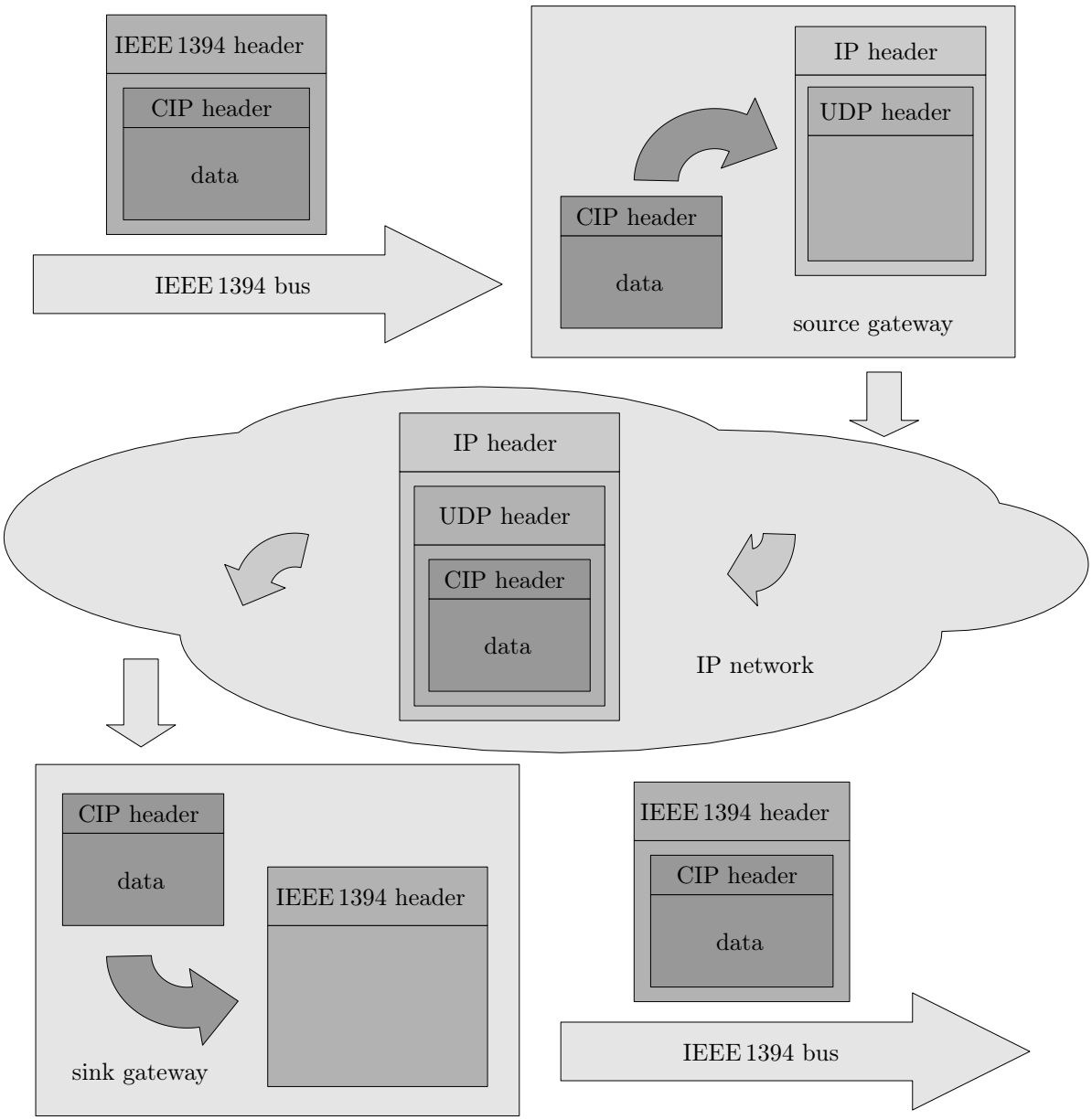

Figure 5.6: Traversal of a packet through encapsulating gateways

does not show the necessary adaptations in the CIP (common isochronous packet according to IEC 61883) header or in the data of the packets, in particular timestamp manipulations. The source gateway receives isochronous packets from the attached IEEE 1394 bus. The IEEE 1394 header and CRC (cyclic redundancy check) are removed because the information inside—like the isochronous channel number—are only useful on an IEEE 1394 bus and IP based protocols like UDP have their own CRC mechanism. Information inside the CIP header has to be maintained. Therefore, the complete CIP is put into a UDP packet. On the sink gateway, the CIP is again put into an isochronous IEEE 1394 packet. Before doing this, adaptations are necessary, in particular the involved timestamps in the CIP header or in the source packet header have to be modified to fit to the IEEE 1394 cycle time on the destination bus. Furthermore,

the SID field (Source Node ID) should be set to the Node ID of the sink gateway.

In essence, this has a similar effect like building a distributed IEEE 1394.1 bridge where the two half-bridges are connected via an IP links. A similar approach is used for wireless IEEE 1394 and wireless IEEE 1394.1 bridges [114]. A difference is that for wireless IEEE 1394, the IEEE 1394 packets are not transmitted on an IP network, but directly on IEEE 802.11 (or one of its variants) or another link layer network. So the encapsulation does not happen on the IP layer, but one layer lower, on data link layer. But it should be mentioned that an encapsulating gateway as outlined here is not the same as a distributed IEEE 1394.1 bridge, although it provides similar functionality to some applications. For example, it does not allow sending arbitrary asynchronous requests to arbitrary nodes on the other bus, there is no global addressing, it just performs dedicated functionalities necessary for multimedia streaming. Furthermore, a real IEEE 1394.1 compliant bridge also synchronises clocks, the IEEE 1394 cycle time, between both IEEE 1394 buses. To be exact, the bridge actually does not synchronise the complete cycle times, it just locks the cycle offsets. Cycle count and second count will differ, but the duration of isochronous cycles is guaranteed do be exactly the same on all bridged buses. The difference between cycle counts on two buses stays constant. In case of a tunnel consisting just of two separate gateways, those clocks and, consequently, the isochronous cycles will drift. This is a real problem that has to be solved. Multimedia streams on IEEE 1394 according to IEC 61883 rely on timestamps included in the packet header[2]. Since IEEE 1394.1 bridges synchronise cycle times (the cycle offset is phase locked), they only have to add a constant delay to these headers. If there is no synchronisation between the cycle times on the source and on the destination network, the timestamps will actually refer to different time bases on the various buses. Since the duration of the isochronous cycles is different on two buses and exactly one isochronous packet has to be provided per cycle, it is not possible to just transmit each received capsuled isochronous packet within one cycle on the sink network. If the cycle time clock on the sink network goes faster than on the source network, the gateway at the sink network will experience buffer underruns, it will not receive enough packets to send one every isochronous cycle. On the other hand, if this clock goes slower than the one at the source network, it will experience buffer overruns, it will receive too many packets to send one every isochronous cycle. In the first case, it has insert empty packets or split packets from time to time. In the latter case it has to drop packets or to combine packets. IEEE 1394 requires an accuracy of the cycle timer of 100 ppm. This means that in the extreme case with two very inaccurate clocks, one at +100 ppm, one at −100 ppm, a deviation of one complete isochronous cycle could happen every 5000 cycles (every 625 ms). So it can be necessary for the sink gateway to perform such adaptations more than once in a second. Therefore, the gateways have to use a more sophisticated algorithm to adapt the timestamps for each bus. They have to observe the drift between the cycle times and also have to re-packetise some isochronous packets in order to make sure that exactly one isochronous packet is sent during each isochronous cycle. And this re-packetising is not as trivial as it might look at the first sight. The easiest case is to insert empty packets in case of buffer underruns. In case of buffer overruns, it is not possible to just combine two original packets. This could violate the limit of reserved bandwidth. Furthermore, if a data format is used that requires timestamps in the SYT field (e. g. audio data according to IEC 61883-6), there will be a SYT interval, which limits the maximum number of samples in one isochronous packet. Therefore, it is necessary to distribute the excessive data smoothly

[2] Depending on the exact data type and format, this could be a timestamp in the CIP header or in the source packet header.

over more packets.

The other way round is even easier: Since IP can be carried in IEEE 1394 packets [67, 68], IP routers can be used to couple IP networks via IEEE 1394. This can be an ordinary PC with one Ethernet interface and one IEEE 1394 interface. It runs the IP protocol on both interfaces and provides routing functionality between them. As mentioned above, another proper solution is transmitting the RTP stream as asynchronous stream on IEEE 1394.

This tunnelling approach is, of course, only possible, if the intermediate network fulfils the bandwidth requirements. Since the data formats usually used on IEEE 1394 require much more bandwidth than the data formats usually used on the Internet, it might not be possible to transfer all the data via some IP networks. In this case, some transcoding would be necessary making the tunnelling approach useless.

Because of these deficiencies, an encapsulating gateway is not considered as a proper solution for coupling networks for real-time multimedia streaming. In fact, it is the worst of all three presented approaches. Therefore, this thesis proposes the implementation of other kinds of gateway, in particular the data format layer gateway.

5.1.3 Transport Layer Gateway

A third approach is a mixture between the above-mentioned solutions. On each network the usual packet format is used, i. e. RTP on IP and CIP (common isochronous packet according to IEC 61883 [58]) on IEEE 1394. But within those packets, media data is coded as it was in the source network, which might be MPEG, DV, RealAudio etc. Such solution is described in [109], where DV (which is common on IEEE 1394, but unusual in IP networks) is transported in IP networks in form of RTP packets.

A prerequisite for such gateway architecture is that the transport protocol on the sink network supports the data format used on the source network. As described in Section 3.2, each transport protocol has a set of data formats it supports. For example, on IEEE 1394 it is possible to transmit audio as MPEG audio in MPEG-2 transport streams [63] or as raw audio PCM samples [62] in common isochronous packets according to IEC 61883. However, it is not possible to transport RealAudio in these packets. Other kinds of network, e. g. MOST, are even more restrictive with regard to supported data formats. On the other hand, in IP networks there is more flexibility, there exist many RFCs for transferring various data formats over RTP. Nevertheless, although many data formats can be transferred via RTP, this does not necessarily mean that typical devices or applications can handle them. Some data formats have to be regarded as exotic, although their transmission is standardised and, therefore, it is possible to build a transport layer gateway for stream using this data format.

This approach avoids the considerable computational effort usually necessary to transcode a stream from one data format to another. Furthermore, a decrease of quality, which is often encountered when transcoding to some lossy data format, is avoided. The gateway does not have to transform the media data itself, it is just passed through. For handling of the transport protocol exactly the same applies as for the data format layer gateway, in particular what is written there about handling of timestamps. The architecture of such gateway can be very similar to that of a data format layer gateway (see Figure 5.1). It also consists of input and output modules, but these modules can be much simpler. The input modules do not have decoders, the output modules do not have encoders. There is no kind of intermediate “raw” data format, the format of the data handed over from the input to the output module is the same that is used for transmission on the source network. Because of this lack of a generic

intermediate format, there are more restrictions in possible combinations between input and output modules since only modules responsible for the same data formats can interact with each other. Although the linking between input and output modules is rather static and not as dynamic as in data format layer gateways, it still makes sense to preserve the architecture consisting of input and output modules instead of combining input and output functionality into one monolithic streaming module. This approach preserves flexibility, as it allows easy addition of support for additional kinds of network by simply adding the corresponding input or output modules. Moreover, it enables an upgrade path to smoothly extend the gateway to a data format layer gateway by adding decoders and encoders to the input and output modules stepwise and, thus, adding support for generic intermediate formats. This would make the combination of input and output modules for handling a stream more dynamic.

Concerning the number of necessary input and output modules in a transport layer gateway, it might give the impression that there are fewer different modules necessary than in a data format layer gateway because fewer different formats are supported. All data formats that are only supported on one network are excluded. This impression is not true because, in order to use the same data format on both networks, the gateway has to support data formats that are exotic and not commonly used on one of the networks. A data format layer gateway would probably not support such exotic data formats, but only the common ones on all networks. The number of input and output modules depends more on the completeness of the gateway implementation than on the type. If a data format layer gateway is really complete, which means that it supports all data formats that are possible on all involved networks, it will, of course, comprise many more input and output modules than a transport layer gateway. But typically a data format layer gateway will only support the most common data formats for each network.

The transport layer gateway encounters the same problem as the encapsulating approach: The data format might not fit well the features like QoS parameters, in particular the available bandwidth, of the sink network. This is one more reason, in addition to data format restrictions, that this solution will not be possible on each combination of networks.

Opposed to encapsulating, there might be devices which are able to handle streams in the uncommon data format using the common transport protocol, provided that the data format is not too uncommon. Nevertheless, some devices will probably be excluded. Therefore, and because this approach is not very general and not possible for many combinations of data format and transport protocol, a data format layer gateway is preferable because it is the most universal kind of gateway.

5.1.4 Media Synchronisation Concept

As already mentioned, an important feature of the transport protocols for multimedia streaming is the synchronisation of clocks at source and sink devices [123]. For this purpose, some of the media data units are tagged with timestamps (often called "presentation timestamps" indicating that they mark the time the corresponding unit is intended to be presented at the sink). Depending on the involved protocol, these timestamps are applied to different data units, for example to complete audio frames or to single audio samples.

These timestamps can directly refer to a time base that is synchronised by means of another protocol between source and sink, as it is done on IEEE 1394. There the IEEE 1394 cycle time, which is synchronised automatically between all nodes on a bus, is used as a reference time base. The deviation of the reference time between the nodes can be determined by the

propagation delay from the cycle master, it will not drift. The packets transmitted according to IEC 61883 can have timestamps at two positions. There is a synchronisation timestamp (the `SYT` field) in the CIP header. This mechanism is used for example for audio data according to IEC 61883-6 [62]. This field is available in every isochronous packet. Usually, this field is not used in every isochronous packet, and if it is used, it usually does not refer to the first sample (or other atomic data unit) of the packet. Instead, depending on the data format and some other parameters (like nominal sample frequency), a `SYT` interval is defined, which determines the number of samples between two that are related to a timestamp. For example, if the `SYT` interval is 4, every fourth sample will correspond to such synchronisation timestamp. Since at most one `SYT` is available per isochronous packet, the `SYT` interval also limits the number of samples that can be transported within one isochronous packet. Therefore, the `SYT` interval is selected slightly above the average number of samples per isochronous cycle. The algorithm to determine the `SYT` interval is specified in the appropriate standard of the series of IEC 61883. If the `SYT` is used, it will refer to the lower 16 bits of the cycle time, so it covers all 12 bits of the cycle offset and 4 bits of the cycle count. This means that the `SYT` field wraps around every 16 cycles (every 2 ms). This, furthermore, has the consequence that these timestamps cannot point more than 16 cycles (2 ms) into the future. Usually, a much lower offset in the range of 2 or 3 cycles (several 100 µs) is used. The other possibility to store timestamps in CIP packets according to IEC 61883 is the source packet header (SPH). This mechanism is used, for example, to transmit MPEG-2 transport streams according to IEC 61883-4 [63]. Here every source packet is tagged with a timestamp. Depending on the data rate and the source packet size, it is possible to have several timestamps per isochronous packet or to have several isochronous packets between timestamps. If this timestamp in the source packet header is used, it will correspond to the lower 25 bits of the cycle time. Therefore, it covers the complete cycle offset and the complete cycle count. So these timestamps wrap around every 8000 cycles (i. e. every second) and, therefore, can point much more into the future than `SYT` timestamps. In practise the timestamps usually point just a few isochronous cycles into the future. These timestamps in the source packet header indicate the time at that the transport layer should forward the corresponding packet to the higher layers at the receiver.

In RTP, the timestamps do not refer to a global clock. Timestamps are transmitted in every RTP packet and refer to the first octet in the RTP data packet. The timestamp is 32 bit long, but the temporal resolution is not fixed, it depends on the data format, some data formats require a finer resolution, some need a coarser one. Since there is no global time base, RTCP sender reports (SR) are used to transmit a mapping between the RTP timestamps and wallclock time. But since that wallclock time is usually not synchronised between the devices, these timestamps are not useful for the intra-media synchronisation described here. They are really useful for inter-media synchronisation, but for intra-media synchronisation other means are necessary. A common approach is to use a local clock and calculate an offset for the RTP timestamps that leads to appropriately filled buffers in the receiver. Since this local clock at the sink is not synchronised with the source and, therefore, will drift away, the offset must be adapted. Usually, when deviation is too high and leads to a buffer overrun or a buffer underrun at the sink, a re-synchronisation is initiated, which often leads to perceivable disruptions. An example for another possible mechanism is the one used in the gateway for audio data between IP and IEEE 1394 in Section 6.2 [61]. Here the arrival rate of audio data and the buffer usage are observed, and the output sample rate is adapted accordingly to keep the buffer usage around a certain level. In this case, the timestamps are not used at all as long as there is only one stream, i. e. only intra-media synchronisation is performed. This alone

is not sufficient if there are several streams requiring inter-stream synchronisation. In this case, the RTP timestamps have to be used in essence to determine the required relation of the buffer usage inside the gateway for these streams. Given that intra-media synchronisation for one stream can be maintained using the algorithm described above, the author of the present dissertation proposes the following strategy to enforce inter-media synchronisation with other streams and implicitly provide intra-stream synchronisation for these other streams [123]:

1. Select one stream as reference. A good choice is to select an audio stream because timing of audio is more critical than video timing. Jitter in an audio stream is detected more easily by a user than in a video stream, and users are much more annoyed by inserted or deleted blocks of audio than in case of such modifications in video [106].

2. For a data unit of the reference stream tagged with a timestamp, calculate the corresponding wallclock time $t_{\mathrm{a,ref}}$ (using the mapping obtained in the last sender report).

3. Calculate the actual presentation time $t_{\mathrm{a,act}}$ of this data unit at the base of a local reference clock (wallclock time) taking into account the current buffer usage and the real (not nominal) sample or frame rate. This is the time the timestamp on the sink network will refer to.

4. Calculate the difference $t_{\mathrm{diff}} = t_{\mathrm{a,act}} - t_{\mathrm{a,ref}}$ between these two times to get a measure for the difference between the local time base and the time base at the source device[3].

5. For a data unit of another stream with a timestamp, calculate the corresponding wallclock time $t_{\mathrm{b,ref}}$ (using the mapping obtained in the last sender report).

6. Calculate the target presentation time $t_{\mathrm{b,tar}} = t_{\mathrm{b,ref}} + t_{\mathrm{diff}}$ at the base of the local clock.

If the local reference clock does not use wallclock time as reference, it has to be converted. Basically, the gateway has to determine for each data unit (sample, frame, etc.) the target presentation time on the sink network (using the corresponding time base on the sink network). According to these presentation times, the data units have to be grouped to packets, tagged with timestamps and scheduled for transmission according to the rules of the transport protocol on the sink network. The buffer usage for another than the reference stream depends on the relationship of its timestamps to the ones of the reference stream.

Although the mechanism described above is adapted specifically for a gateway with several RTP streams at the input, it is just a special version of the general synchronisation concept, which is visualised in Figure 5.7. The gateway has some local reference clock. In the example above, this reference clock is implicitly defined by the arrival rate of an RTP stream in conjunction with its mapping between media presentation timestamps and wallclock time. The input modules of the gateway tag media data with timestamps of this time base (1). For this purpose, they usually have to convert incoming timestamps to this time base, unless the media timestamps already refer to this time base as the reference stream in the example above. Media data tagged with timestamps is handed over to the appropriate output module (2). Since all media data in the stream converter use the same time base as reference, inter-media synchronisation within the gateway is preserved. The output modules again have to convert this local timestamps to the time base used on the sink network (3) and can use the local reference clock to schedule packets for transmission appropriately (4). The figure is a bit simplified as it

[3] If the clocks were synchronised, this difference would be constant.

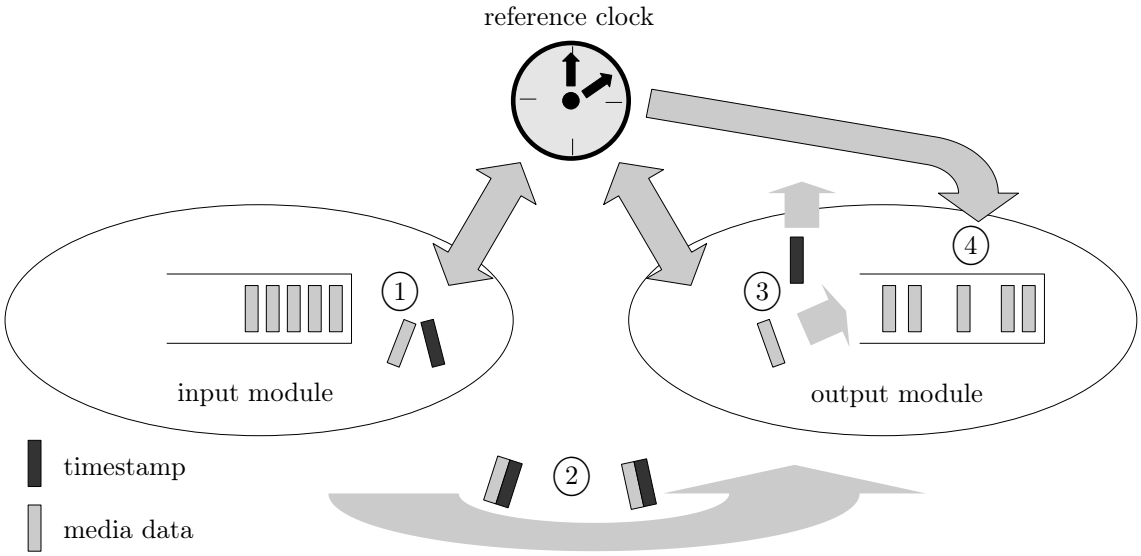

Figure 5.7: Visualisation of the synchronisation concept

does not show manipulations of media data, it just illustrates the timestamp handling within the gateway. The timestamps of the transport protocols are not directly visualised because different protocols use different places and different frequencies for those timestamps.

In MOST, there are no transport layer timestamps because MOST is a synchronous bus, where all sampling clocks in the network are synchronised with data transmission. Multimedia data is delivered exactly in time, buffers at the sink devices are almost completely avoided. A consequence is that in gateways with MOST networks as targets the gateway has to do all buffering that is necessary to balance the unsteadiness of the incoming stream, it cannot leave this task to the sink device.

In addition to timestamps on the transport layer, there are sometimes also timestamps within the media data. For example, in MPEG-2 transport streams and program streams, there are additional synchronisation timestamps. They are needed because these streams are multiplexed streams with several media streams integrated that need inter-media synchronisation. On the one hand, there are timestamps in the stream that define the reference time base. In case of transport streams, they are called PCR (program clock reference). In the case of program streams, they are called SCR (system clock reference). On the other hand, media data is tagged with presentation timestamps that indicate the presentation time referring to the reference time. Therefore, when an MPEG-2 transport stream is transferred over IEEE 1394 according to IEC 61883-4, several layers of timestamps are involved. First, the delivery of the MPEG-2 transport stream packets to the MPEG-2 demultiplexer has to be performed at exact points in time. Otherwise, the other, MPEG-2 internal, synchronisation mechanisms would not work. For this purpose, timestamps in the source packet header are used to eliminate the transport jitter caused by the isochronous transport over IEEE 1394. The demultiplexer uses the PCRs (or SCRs) to set its internal reference clock. Therefore, it is necessary that it gets the source packets at exactly the correct times when the PCRs (or SCRs) are intended to be valid. After decoding, the video and audio decoders use presentation timestamps to synchronise presentation with the reference clock. If a gateway relays the complete transport stream, it does not demultiplex it and only has to deal with the timestamps on the transport protocol layer, not with the internal

timestamps of the MPEG-2 transport streams, this is deferred to the sink device. However, if the gateway demultiplexes such streams and transmits, for example, audio and video as separate streams on the sink network, it will have to deal with these timestamps. To calculate the correct presentation timestamps, which it needs to tag the media streams in the transport protocol on the sink network, the author of this dissertation proposes the following algorithm:

1. Use the timestamps of the incoming transport protocol to determine the correct times when the packets are valid.

2. In packets containing a PCR (or SCR), these timestamps together with the transport protocol timestamp are used to get a mapping between the MPEG-2 reference clock and the transport layer clock. Such algorithm can be found in [124].

3. When a media presentation timestamp is encountered, the mapping obtained in the previous step is used to convert this timestamp to the time base of the transport protocol.

4. This timestamp at the time base of the source transport protocol has to be converted to the time base of the sink network transport protocol.

It does not need to use any additional local clock as time base. The only clock needed is the one used for the transport layer reception or transmission.

5.2 Control

The gateway must, of course, also map the mechanisms for setting up a connection between the two networks. This means the use of the connection management procedures specified in IEC 61883-1 [58] on IEEE 1394 and the use of RTSP on IP. On MOST, there is no protocol with a special name for connection management, but there is a set of functions that have to be called at source and sink in order to get connections established. The gateway should also provide a translation of the higher layer protocols like HAVi, Jini, etc. The control task is not the main focus of the present work, which concentrates on the streaming task. There is quite some existing work on the area of control gateways for multimedia home networks, which also was mentioned in Section 4.2 about related work. Nevertheless, the control task is an essential complement to the streaming part.

Since for some protocols, the mechanisms for establishing and maintaining the streaming connection are tightly coupled with the actual transport of data, for example when transport according to IEC 61883 on IEEE 1394 is used, this subtask of the control part might actually be handled by the input and output modules within the gateway. In this case, there would, of course, be a stronger interaction between the control and the streaming part, in particular regarding these modules. However, in this section, the control task is treated as one whole part, although in the actual implementation it might be split between the control part and the streaming part.

The mapping between control protocols is a rather complex task. The main problem is that different protocols may have completely different views of the network. While one protocol might see the network as a collection of devices consisting of several functional components with a certain relationship between a device and its functional components, another one might not have such structural view and just consider the network as an accumulation of services. An example of the first are both major protocols on IEEE 1394, AV/C and HAVi. At a first layer

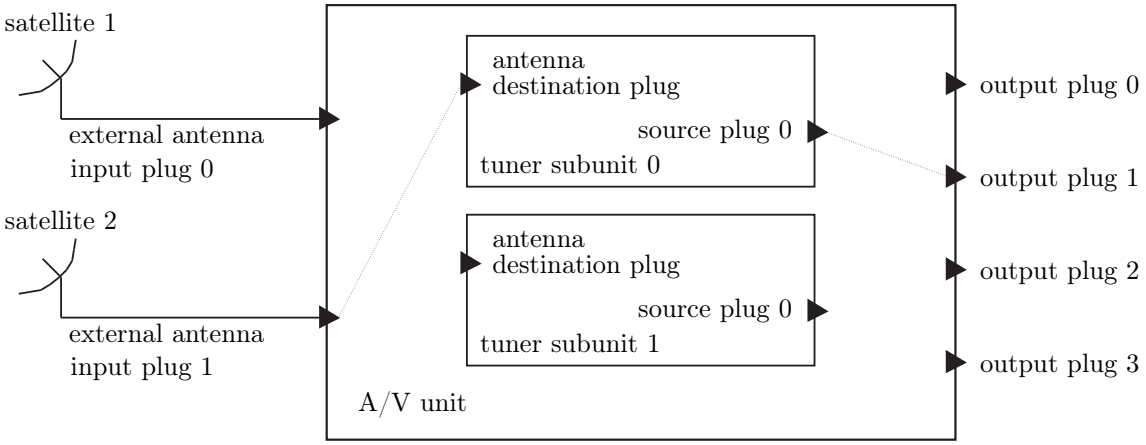

Figure 5.8: Concept of (virtual) plugs used in protocols on IEEE 1394

they see units in AV/C and DCMs (device control modules) in HAVi, which basically comprise the devices connected to the bus. These can consist of several function blocks, which are called subunits in AV/C and FCM (functional component module) in HAVi. And these function blocks, like CD player, audio amplifier, satellite tuner, offer several functions. MOST uses a similar hierarchical addressing scheme. Protocols on IP are usually less hierarchical. Another example for a concept that is not available on all networks is the concept of (virtual) plugs, which is present in both major control protocols on IEEE 1394; a similar concept is unknown in the IP world. Figure 5.8 demonstrates this concept and shows connections between the plugs of the functional units (subunits in case of AV/C, in case of HAVi the same entities would be called FCMs) and the external plugs of the unit. In this example, there is one connection from the second external antenna input plug (with the number 1) of the unit to the antenna destination plug of tuner subunit 1. The tuner subunits always have only one antenna destination plug, but they might provide several source plugs, although the tuners in the example in Figure 5.8 both have one single source plug, labelled with the number 0. From this single source plug of tuner subunit 1, there is a connection to the second output plug (number 1) of the unit. These connections are established by special commands of either AV/C or HAVi addressed to the unit or DCM, respectively, and define to which functional unit the external inputs and outputs are routed. It has to be stressed that these connections are completely virtual, they are a comprehensible equivalent to cable connections in the analogue world, but here no physical plugs exist. Figure 5.9 demonstrates isochronous connections between the external plugs of different devices. These connections are established and maintained using the connection management procedures of IEC 61883-1 [58], the common foundation of AV/C and HAVi. These connections are purely virtual, too. The plugs are controlled using plug control registers (PCRs), which are divided into input plug control registers (iPCRs) and output plug control registers (oPCRs), and do not have anything to do with physical IEEE 1394 plugs. The number of PCRs a device provides indicates how many incoming and outgoing isochronous streams it can handle at the same time.

Another issue, besides the addressing matters, is that there might be no direct mapping between functions in both protocols. For instance, there might be a command to tune a tuner to a certain frequency and a set of functions to actually start streaming on one network or protocol, respectively, whereas on the other one, there might be a command that tunes and

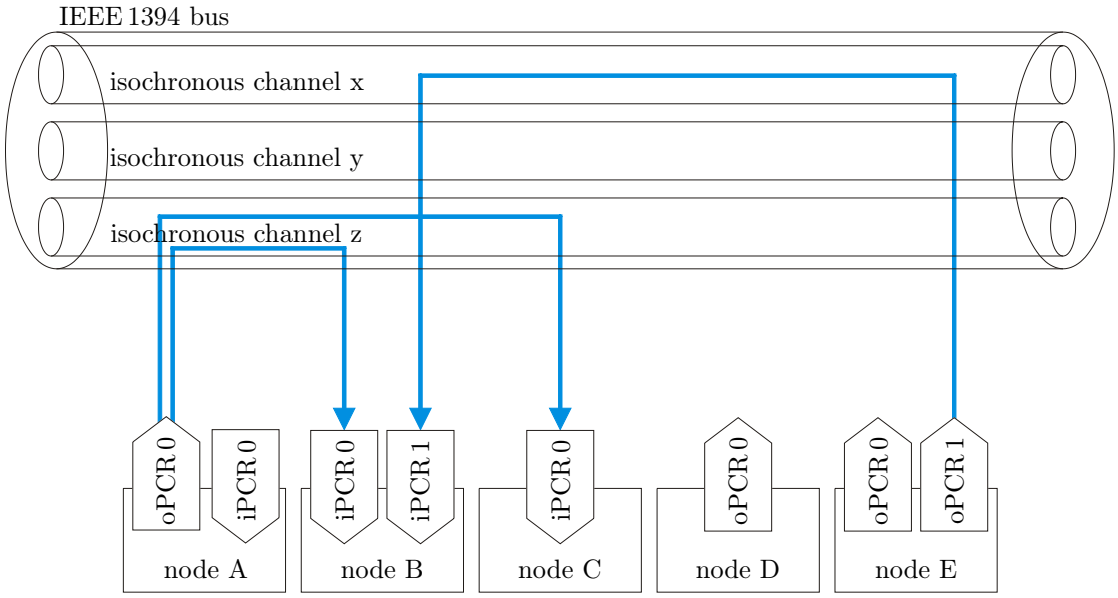

Figure 5.9: Point-to-point connections according to IEC 61883-1

sets up the stream at once.

Therefore, it is rather difficult to design a universal gateway, which can be extended by plugins for various protocols, because this would require a very abstract view of the control tasks. A more viable way is to implement separate gateways for each pair of control protocol. Implementation of some gateway functionality might be omitted if interoperability functionality inherently exists in another protocol that is handled by the gateway. For example, HAVi provides means to control AV/C devices (and even devices using other, proprietary protocols) by using DCM code units [60]. Therefore, if there are AV/C and HAVi devices on an IEEE 1394 bus on the one side of the gateway, it might be sufficient for the gateway to just operate the HAVi protocol. The AV/C devices would then controlled by a HAVi device[4], which provides the functionality of the AV/C devices to the HAVi network and acts as a kind of proxy.

For each network, the gateway proposed in the present thesis represents devices, services, software elements (or whatever the name of these entities is in the protocol used on one network) of one network by appropriate proxies in the other network. As mentioned before, this mapping is not easy and can sometimes not be achieved directly, in particular if the view of the network differs too much between the protocols used on those networks. Some examples for possible mappings are given here. One example is the mapping between an IEEE 1394 bus containing HAVi devices and an IEEE 1394 bus containing AV/C devices, which is shown in Figure 5.10. This example is not very realistic, as it just couples two buses of the same network technology, but it is illustrative for addressing issues. On the HAVi side, all AV/C devices (units) of the other side can be represented by DCMs which are hosted by the gateway device. This means that the software elements of several DCMs are located within one device, the gateway. HAVi allows this and uses this mechanism implicitly to make functionality of non-HAVi devices—device classes LAV (legacy A/V device) and BAV (base A/V device) in HAVi terminology—

[4]This must be a device of a high device class like IAV (intermediate A/V device) or FAV (full A/V device) [60], and a DCM code unit must be available.

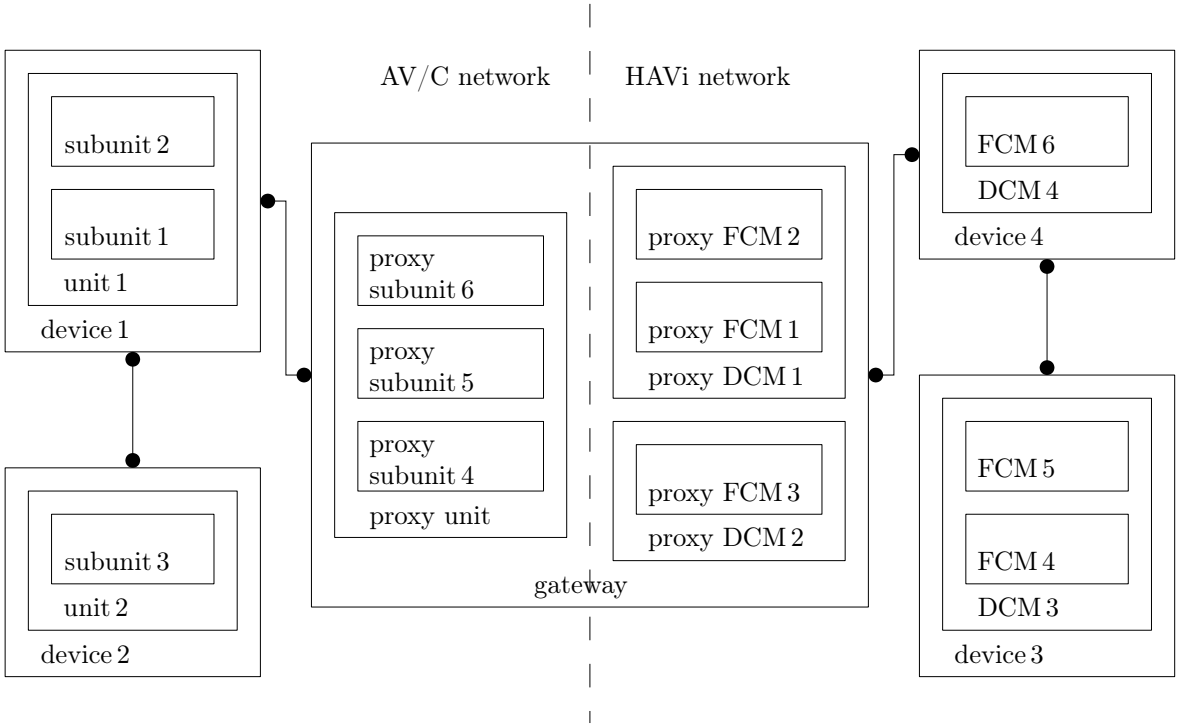

Figure 5.10: Proxy elements in gateway between AV/C and HAVi

available by hosting DCMs in other HAVi devices. The software elements for the functional components are, of course, also located in the gateway device. Therefore, commands sent to those proxy software elements can, after a translation of the functionality to AV/C, be relayed directly to the corresponding unit or subunit on the AV/C side. The mapping in the other direction is not that easy, although the addressing scheme on HAVi and AV/C is similar. In AV/C, there is a strict mapping between a unit and the device the commands are addressed to. All AV/C commands sent to a particular device targeting some functionality of this unit or one of its subunits. In other words, there can only be one unit within a device. Therefore, on the AV/C side, the complete HAVi network is represented by one single proxy unit containing subunits, which are proxies for all FCMs on the HAVi side. Therefore, commands targeting one proxy subunit can be relayed to the corresponding FCM on the HAVi side. However, commands directed to the unit cannot be directly forwarded to a DCM on HAVi. In cases of commands that affect subunits on different devices one command on the AV/C side might even trigger two commands to two different DCMs on the HAVi side. In particular, if a connection between two of the proxy subunits is established, which seems to be an internal connection between two subunit plugs on the AV/C side, it will have to be assembled of three connections on the HAVi side. There are two connections needed between FCM plugs and DCM plugs as well as an external isochronous connection according to IEC 61883. So the gateway has to issue two `Connect` commands to the two involved DCMs and establish the isochronous connection using the connection management procedures of IEC 61883-1 [58]. Of course, all these steps can fail, and the gateway has to handle these failures gracefully, report proper error conditions to the caller and undo parts of the transactions that already have succeeded. Obviously, the "real" connection on the HAVi side has more possible error conditions than an internal connection,

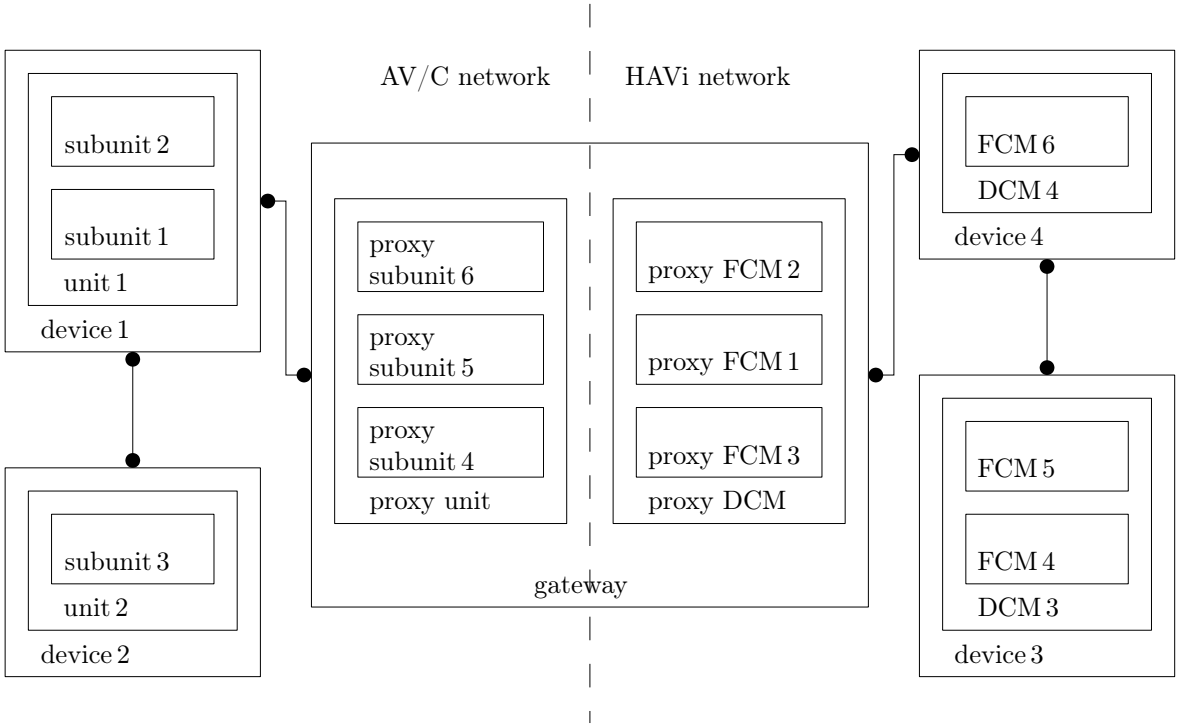

Figure 5.11: Modified setup for proxy elements in gateway between AV/C and HAVi, representing the complete AV/C side as one single proxy DCM

as it looks like on the AV/C side, could have, e. g. bandwidth allocation failures. These error conditions have to be mapped properly to the smaller set of possible error conditions on the AV/C side. Although the proxy configuration on the HAVi side seems to be more flexible and allows a more accurate reproduction of the AV/C side to the HAVi network, this design turns out to cause severe problems when isochronous connections are to be established. Since the HAVi devices are aware that they have to establish a connection between different DCMs, the HAVi stream manager will try to establish an isochronous connection according to IEC 61883. For this purpose, it will query the GUIDs (global unique identifier) of the involved devices. Using this identifier, it will try to find out the current node ID[5], which is needed to issue the necessary requests. This lookup will fail because the involved GUIDs do not exist on the local bus, and it is not possible to send packets to devices on a remote bus on the IEEE 1394 layer. Therefore, even on the HAVi side, it is better to instantiate only one DCM in the gateway that represents all functions available on the remote bus, although an accurate mapping would be possible. This is illustrated in Figure 5.11. It makes the translation of the protocol a bit more complicated because, as mentioned above, the mapping between proxy DCM and target device on the remote bus is not direct and commands addressed to the one and only proxy DCM might trigger interactions with several units and, therefore, devices on the AV/C network. In this scenario, all connections between proxy FCMs for the other bus look like internal connections within one DCM. Hence, the HAVi side will never try to establish external

[5]The GUID is a unique identifier, which never changes and uniquely identifies a device. However, this identifier cannot be used for addressing requests on IEEE 1394. For this purpose, the node ID is used, which can change and, therefore, is not constant for a given device.

isochronous connections according to IEC 61883. Instead, it will issue `Connect` commands to the proxy DCM to establish internal connections, and the gateway will translate it to the appropriate actions on the other bus. If a connection between devices on both sides of the gateway is to be established, the caller will create an internal connection between a proxy FCM plug and a proxy DCM plug, which has to be translated to an internal connection and an external isochronous connection to the gateway on the remote bus. Furthermore, the caller will create "real" external isochronous connection from the gateway to a device on the local bus and an internal connection within this device. The conclusion is that the mapping outlined in Figure 5.10, which tries to represent the entities of the other side as accurately as possible, is not the best one, it is better to treat the complete remote network as one device as shown in Figure 5.11.

As a side note to the coupling issue between two IEEE 1394 buses, it should be mentioned that a proper means of coupling two IEEE 1394 buses would be a bridge according to IEEE 1394.1 [66]. This bridge would allow direct addressing of devices on the other side. It, furthermore, provides mechanisms for establishing, maintaining, and breaking of isochronous connection between several buses. HAVi devices could directly instantiate DCMs for the AV/C devices. For the other direction, HAVi devices controlled via AV/C, a gateway would still be needed. Unfortunately, HAVi and AV/C were not defined for IEEE 1394 networks containing bridges. Therefore, extensions to the respective standards are necessary. HAVi needs two main extensions. The device detection mechanism must be adapted so that also devices on remote buses are detected and, therefore, visible for HAVi. On the other hand, the stream manager must be modified. It cannot use the connection management procedures of IEC 61883-1 [58] to handle connections where remote devices are involved. Instead, it has to use the mechanisms specified for isochronous connection in bridged environments specified in IEEE 1394.1 [66]. A company has been working on proprietary extensions to HAVi to allow operations in such environment[6], but since HAVi is not actively maintained at the moment, it is questionable whether there will be a new version of the HAVi specification that has bridge support included.

The principles developed above for coupling two IEEE 1394 buses with AV/C and HAVi also apply for coupling between other networks. An example is given in Section 6.2, where a gateway between IEEE 1394 and IP is described. In this case, all devices on the Internet are represented as one unit/DCM on IEEE 1394. In fact, all radio services available in the Internet are represented as one tuner FCM (a tuner subunit in AV/C would, of course, be equivalent). So it is a general principle that the gateway should represent the complete other side as if it was one single device. Cases in which it it is better to make the structure of the remote network transparent are possible, but not the general scenario.

Of course, it is also a design question whether the gateway should be unidirectional or bidirectional. It is possible to design a fully transparent bidirectional gateway, where both sides have a complete view of the services available on the other side (with the restrictions concerning the mapping mentioned above). In the example above, devices of both networks would see all connections existing on the other bus, can drop them or establish new ones (as far as the access right mechanisms of the involved protocols allow it), etc. On the other hand, it is also possible to build a unidirectional gateway where only one side of the gateway sees the services of the other side, but not the other way round. Devices of the other bus might see that they have, for example, connections to the gateway, but they have no view of the devices behind the gateway where the connection really ends. They are not able to actively communicate with

[6] Details about this work are confidential and cannot be given here.

devices on the other side of the network, they cannot establish connection to or between them.

The decision whether the control part of the gateway should be unidirectional or bidirectional is not coupled to the decision whether the streaming part should be unidirectional or bidirectional. For example, it makes sense to have a bidirectional streaming part, even if the control part is unidirectional. In this case devices of the "privileged" network can use devices of the other network as sources and sinks. So local satellite tuners can present their content on remote TV sets, and the program received by remote satellite receivers can be presented at local TV sets. Also the other combination is possible, bidirectional control part and unidirectional streaming part within one gateway. In this case all devices and services can be controlled from both sides, but streaming connections are only possible in one direction. This might be a bit confusing for the user because connections which seem to be possible, cannot be established. Therefore, it is desirable to couple bidirectional control parts also with bidirectional streaming parts. If the streaming part is just unidirectional, it makes sense to design a unidirectional control part that only makes a subset of the services from the other side available, either sinks or sources of multimedia data depending on direction of the streaming part.

An interesting design choice is when streaming on the source network is started. From the network efficiency point of view, it would be the best to start streaming when the streaming connection on the sink network has been established. So there would only be a stream if there was a sink device consuming it. On the other hand, if an application on the sink network selects a service (a TV or radio program) at a tuner proxy or a starts playing a track on a CD player proxy located in the gateway device, it will expect that upon successful completion the corresponding stream is available at this proxy. The devices or applications on the target do not necessarily have the information, that they are talking to a proxy. From their viewpoint they might talk to a real tuner, CD player, etc. Therefore, it seems to be more correct that, when such command (tune, select etc.) is issued to a proxy for the corresponding unit on the source network, the gateway forwards the appropriate command to the source device and establishes the streaming connection on the source network. The input module of the stream converter can then start buffering and, when the streaming connection has been established on the sink network, it can commence immediately.

5.3 Content Protection and Digital Rights Management

It might be necessary that the gateway has to handle streams that are protected by some digital rights management system. In the practical work accompanying the present thesis, no encrypted streams were available. Unfortunately, in-depth information about digital rights management is only available under very restrictive license conditions. Only informational versions of specifications are available freely. The technology itself is also only available under restrictive conditions. Therefore, only some theoretical considerations concerning content protection are given, but there have been no practical experiments or implementations.

A gateway has three possibilities to handle encrypted multimedia stream. One strategy is not to care about content protection and retransmit the stream as it is on the sink network. It is obvious that this approach can only work for encapsulating gateways and transport layer gateways. And for transport layer networks, it is only possible if the used protection system is also available on the sink network. Data format layer gateways have to transcode the multimedia stream into another data format and, consequently, have decrypt them first. If the stream is relayed unmodified, it is obvious that the sink devices have to decrypt themselves. While this

clearly limits the number of possible sink devices that are able to handle the stream, it is probably exactly what content providers want. In order to decrypt the stream, sink devices need some information, in particular cryptographic keys. The way how to get this information depends on the content protection system in use. There might be digital rights management systems where all trusted sink devices have all the keys necessary for decryption integrated. It might also be possible that these keys can be obtained by some means like from the Internet after some kind of registration. So if the sink devices can perform decryption with the information they have inside or they can obtain without intervention of the gateway, the control part of the gateway does not have to care about encryption, besides, perhaps, including the information that the stream is not available freely in its service list. However, there are DRM systems, in which the content keys are changed quite frequently and where key exchanges are necessary between source and sink. DTCP is one example [103]. In this case, the control part of the gateway also gets involved. It has to relay the key exchange messages between source and sink. So for the source, it would look like it would communicate with the gateway, and the gateway would be the sink of the multimedia stream, while for the sink, it would look also like it would communicate with the gateway, and the gateway would be the source of the stream. From the informational specification [103] and the white paper [125], it is not clear whether such approach would work and whether there are any mechanisms preventing such intermediate devices relaying these authentication and key exchange commands. Unfortunately, neither the full specification nor a test environment was available to the author of this thesis. But there is at least one scenario, which seems problematic. If there are two systems on the sink network listening to the encrypted multimedia stream, both will try to negotiate with the source. While it is certainly possible that two devices receive the stream and independently exchange keys with the source, the problem in this case is that, from the point of view of the source, both requests seem to come from the same node, from the gateway device. It is not clear whether the source would allow two independent authentication processes in parallel from the same node. At least for this reason, it is questionable whether this approach would work. Obviously, further investigation is necessary on this topic.

Both other possibilities for the gateway to handle encrypted multimedia streams require the gateway to decrypt the stream. Unless some circumvention technique is used, which is legally forbidden in some countries and the legality of which in other countries is at least questionable, this requires cooperation with the provider of the digital rights management system. One possibility for the gateway is to transmit the unencrypted stream on the sink network. The clear advantages of this approach is that it allows the most flexibility with respect to supported sink devices, no other approach could make the stream available to more sink devices. Therefore, this is definitely what users would like most. Unfortunately, content providers do not like this strategy because they will not allow their previously protected streams to be transmitted unencrypted over a digital network. They might allow it if it was ensured that before retransmission the quality was degraded or if it was somehow ensured that only a limited number of devices or only some classes of devices, in particular no recording devices, would be connected to the sink network. Therefore, it would be hard to get an approval of the provider of the digital rights management system for such gateway device. And without such approval, the gateway would not get access to the cryptographic keys necessary, only perhaps in some way of questionable legality. Since inside the gateway the stream is available unencrypted, the gateway can perform operations on the data format layer, so data format layer networks can be designed. In fact, the encryption could be interpreted as part of the data format, and consequently, each such gateway could be interpreted as data format layer

gateway. Nevertheless, if the gateway does not manipulate the stream in another way than decrypting and possibly encrypting it again, it should still be referred to as transport layer gateway or encapsulating gateway. The term data format layer gateway should stay reserved for gateways that perform substantial operations on the data format layer, in particular decode the data format. The decryption could be integrated into the input modules. Depending on the involved content protection system, adaptations in the control part might be necessary to negotiate keys with the source device. Then there must be an interface between the control part and the streaming part of the gateway to give the streaming part the information necessary for decryption.

The third and last strategy for the gateway is to decrypt the stream received from the source network and encrypt it again before transmission on the sink network. Such gateway could get an approval of the system provider and access to the required cryptographic keys because it is ensured that the stream is never available in unencrypted digital form. Decryption can be integrated into input modules, encryption can be integrated into output modules. The gateway can perform any manipulations on the data format. Obviously, it can only transmit data formats that are supported by the content protection system. In this case, it is even possible to use different content protection systems on both networks. This might be necessary, if the system on the source network does not support the sink network. Then another system has to be employed on the sink network. As for the other two strategies for the gateway, depending on the DRM system in charge, the control part of the gateway might get involved into key exchange procedures. It might even have to ensure certain conditions, e. g. limit the number of sink devices having simultaneous access to the stream. Evidently, there must be an interface between the control part and the streaming part for exchanging the necessary key information.

It is possible for a gateway not to support protected multimedia streams, and all implementations developed during this thesis did not support such streams. At present, there are many free multimedia streams available. But it is not clear, whether in future most multimedia content will be encrypted and, therefore, streaming gateways will be required to support DRM. Consequently, future will tell whether support for DRM is an essential part of a multimedia streaming gateway or rather an optional add-on.

6 Implementations

During work on the present thesis, several examples for multimedia streaming gateways were implemented. None of these gateways reached a maturity like a commercial product. They all have some missing functionality, which must be added to reach a product state. However, they demonstrate the main functionality and serve as a proof of concept or at least as a reasonable emulation for some coupling scenarios. This chapter describes these implementations and discusses their strengths and weaknesses. It also includes some results of the EU funded projects InHoMNet and HOME-PLANET. Furthermore, it is pointed out how these implementations can be improved.

6.1 Gateway between DVB and IEEE 1394

This section describes implementations of gateways between DVB (digital video broadcasting, the European standard for digital television) and IEEE 1394. TV programs distributed via cable, terrestrial antennas, or, as in this case, satellites can, of course, not be seen as a classic network as it is used in home or automotive environments. However, the second part of the gateway, the interface to IEEE 1394, clearly demonstrates an output module for MPEG-2 over IEEE 1394 according to IEC 61883-4. The source part, which delivers the MPEG-2 data at a fixed data rate with equidistant time intervals between MPEG-2 transport stream packets, could also be seen as an imitation of a synchronous bus like MOST, which would have similar data delivery properties. Since MPEG-2 data is taken from the source and transmitted on the sink network as MPEG-2 transport streams and no decoding and encoding is performed, it can be seen as an example of a transport layer gateway. However, although no data transcoding is performed, it has to be stressed that MPEG-2 is one of the common formats to transport A/V data over IEEE 1394. Therefore, the typical disadvantage of transport layer gateways compared to data format layer gateways is not effective here. Since the gateway does some manipulations on the data stream, albeit no transcoding, it is, in fact, a bit more than a simple transport layer gateway.

There are two implementations of such gateway. The first one was developed during the EU project InHoMNet, and the streaming part is completely hardware based [1, 54]. The second one was developed during a diploma thesis, it aimed a software based solution for the same functionality and also tried to address some of the weaknesses of the first solution [126].

6.1.1 Hardware Based Solution

One main objective in the European project InHoMNet was to replace analogue TV cabling by a digital home network based on IEEE 1394. The Institute of Computer Technology of Vienna University of Technology was project coordinator and had the task to develop a device called "digital intelligent external tuner". This device should provide TV services received as digital TV stream via DVB-S from a satellite (like the Astra satellite system) to the IEEE 1394. It contains two satellite tuners in order to make two independent multimedia streams originating from

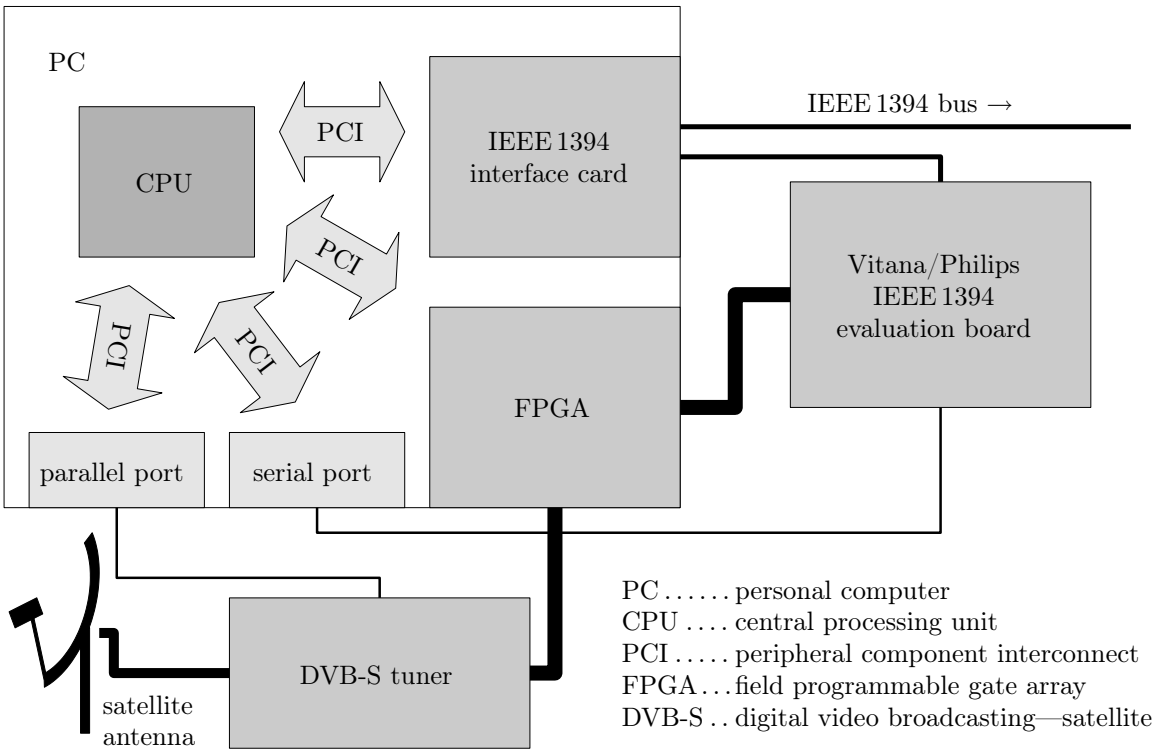

Figure 6.1: Architecture of digital intelligent external tuner from the InHoMNet project

two different transponders available at the same time. A requirement was that it should not transmit complete transport streams, which usually contain several TV and radio programs and need around 40 Mbit/s bandwidth, but it should filter the stream and only transmit a partial MPEG-2 transport stream according to [41], which contains only the TV or radio programs requested by the user and, therefore, significantly reduces the required bandwidth (to a range around 5 Mbit/s for a single TV program). Since the timing requirements of MPEG-2 transport streams are very rigid, as already lined out in Subsection 5.1.4 and discussed more in detail in Subsection 6.1.2, it is very important that the modifications applied to the transport stream in order to generate a partial transport stream do not change the timing of the packets that are passed through. In particular, packets containing PCRs must have a constant delay on their way through the complete system. Otherwise, these timestamps would have to be rewritten, and such calculation of new timestamps should be avoided. The architecture of the external tuner is shown in Figure 6.1. This figure is slightly simplified. It only shows one tuner, whereas the actual system contains two identical tuners. Furthermore, some coupling hardware, like signal level converters, is omitted.

For cost and development speed reasons, a PC based solution was chosen, but provisions were made for a later port to an embedded system. The CPU (the main processor) communicates with the other components via the PCI bus. The PC contains a PCI card manufactured by Nallatech equipped with a XILINX Virtex 800 FPGA, which is used to modify the MPEG-2 transport stream as it will be described later. It contains an IEEE 1394 interface card that is used for control purposes, but not for isochronous transfers. Outside the PC, there are two

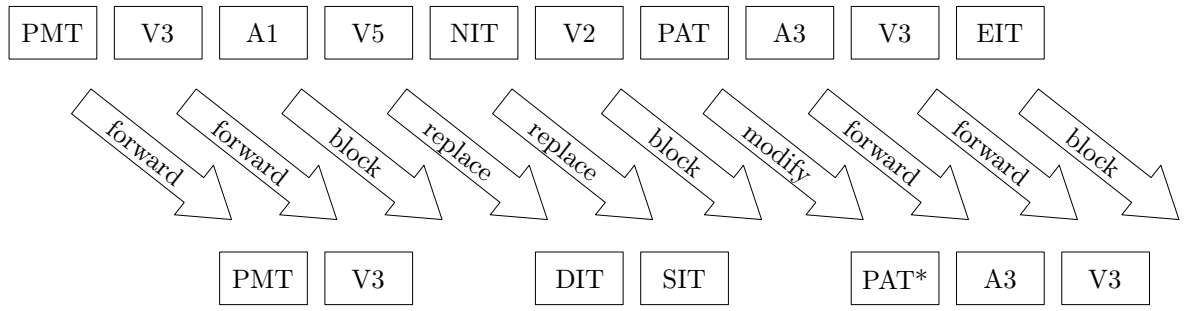

Figure 6.2: Modifications by logic inside FPGA on full MPEG-2 transport stream transforming it into a partial MPEG-2 transport stream (labels starting with "A" denote packets of an audio stream, labels starting with "V" mark video packets, all other packets contain PSI and SI tables)

major components. One is an IEEE 1394 evaluation board by Philips/Vitana[1]. The main feature of this board is that it provides a parallel digital A/V interface and can either output a data stream there received from IEEE 1394 according to IEC 61883 or transmit a data stream received via this A/V interface as isochronous stream on IEEE 1394 [127, 128]. It supports several types of data streams, among them MPEG-2 transport streams. The task of these boards within this setup is to take an MPEG-2 transport stream from the A/V interface, generate timestamps for the source packet header, packetise them according to IEC 61883-4, and transmit them over IEEE 1394. The other external component is a DVB-S satellite tuner. This tuner is connected to a satellite antenna via a coaxial cable. It provides the DVB data stream, an MPEG-2 transport stream, as parallel signal at its output. This parallel output is connected to the FPGA, which modifies the data stream, and the output of the FPGA is connected to the IEEE 1394 evaluation board. The tuner can be controlled over an I^2C bus, which is mainly used to set reception parameters like frequency, polarisation, etc. For this purpose, the tuner is connected via an I^2C converter to the parallel port of the PC. The IEEE 1394 evaluation board is set up and controlled over a serial cable connected to the serial port of the PC.

The path of the multimedia stream starts as analogue signal that is received by the tuner. The tuner converts it into a digital signal and forwards it over a digital 8 bit parallel cable. At this point, it is data of a constant bit rate, the system was designed for a symbol rate of up to 10 MBd, which corresponds to a bit rate of 80 Mbit/s. The actual bit rate varies from transponder to transponder. This stream goes through the FPGA. The logic inside the FPGA transforms the full transport stream into a partial transport stream. Its main tasks are to filter, store, and insert MPEG-2 transport stream packets as shown in Figure 6.2. It has to block all packets belonging to TV programs or services not of interest and to pass through packets of interest unmodified. These packets experience a constant delay of two packets, which is completely independent from the surrounding packets, which might either be blocked or passed through. This latency of two packets is caused by the design of FPGA logic. For this purpose of blocking packets, it can be configured to filter packets according to their PID (a packet identifier that occurs at the same position in every packet). In addition, it has to store information from PSI (program specific information) and SI (service information) tables

[1] Philips Semiconductors PDI1994L11/P11 Firewire Reference Design Kit

[41, 49], which are then read out by the CPU. And it has to insert new packets that are provided by the CPU. Since the timing of packets passed through must not be affected, packets can only be inserted, when packets from the input stream are blocked. Insertion can either be triggered by a timer or by a special incoming packet, which is to be replaced or modified, or by the CPU. The logic inside the FPGA has to insert a modified PAT (program association table), which only contains information about the services of interest. It passes through the unmodified PMTs (program map tables) of the programs used (because programs are always either passed through completely or not at all). It has to insert a new DIT (discontinuity information table) and SIT (selection information table). All other SI tables are blocked. This is necessary to comply with the requirements for valid partial MPEG-2 transport streams [41]. At the output of the FPGA, the data stream has no more a constant data rate because parts of the original stream are chopped out, and the parts that are passed through or replaced maintain their original timing. So there are periods without data and periods with the original rate. This stream is fed into the IEEE 1394 evaluation board, which acts as output module and generates timestamps for the source packet header according to their arrival times at the board. Since the original timing is preserved, the difference between the timestamps of two packets is the same as if the complete transport stream with the filtered packets left in-between would be transmitted, there is no kind of smoothing these intervals. Then the stream exits the evaluation boards as isochronous stream according to IEC 61883-4 [63].

For control purposes, the multimedia middleware HAVi is used [60]. The CPU of the PC runs a HAVi stack implemented in the programming language C. It communicates with other devices via its IEEE 1394 interface card, which is a PCI card manufactured by Unibrain (it is called "FireBoard 400") based on the IEEE 1394 link chip TSB12LV21BPGF (PCILynx) by Texas Instruments. The software runs under the operating system Microsoft Windows 2000, but it was designed in a modular way with two abstraction layers. There is one abstraction layer for operating system dependent functionality like thread synchronisation functions. This abstraction layer was implemented for the Win32 API (application programming interface)—and therefore, works on various variants of Microsoft Windows—and on the Posix threads API, which is used on Linux and many other similar systems [129]. The second abstraction layer is used for access to IEEE 1394. This abstraction layer was fully implemented for Unibrain FireAPI [130], which is available for several version of Microsoft Windows, and it was implemented for the IEEE 1394 library `libraw1394` on Linux [131]. There have also been some rudimentary implementations for other IEEE 1394 APIs. Since the rest of the HAVi stack was written in plain ANSI C [132], it is quite portable, and in fact, this software suite runs successfully at least on Linux and several variants of Microsoft Windows.

This setup has two connections to the IEEE 1394 bus. One is the PCI IEEE 1394 interface card inside the PC that is used for asynchronous traffic of the HAVi protocol. The other one is the IEEE 1394 evaluation board, which only performs isochronous traffic. So there are really two separate nodes with different node IDs on the bus, and isochronous traffic originates from another node ID than asynchronous traffic. It might be an a bit unexpected behaviour and a slightly unclean solution that isochronous traffic has another node ID in its header than the node that is addressed for setup and maintenance of the isochronous connection in the connection management procedures according to IEC 61883-1. However, it does not cause any problems, at least in the environments this setup was tested in.

This solution provides a very clear separation between the control and the streaming part, they are performed by different hardware and use totally different data paths. The streaming part is coupled with the control part via three interfaces:

1. A parallel cable and a converter to I^2C from the parallel port to the tuner is used to control the tuner[2].

2. The interface between the FPGA and the CPU is the PCI bus because the FPGA is placed on a PCI card.

3. There is a serial cable from a serial port of the PC to the IEEE 1394 evaluation board.

The setup worked fine within the InHoMNet project. The project had a very good performance at its final review. It was demonstrated that it this intelligent digital external tuner operates in an environment consisting of several nodes on an IEEE 1394 bus, among them several sinks for MPEG-2 transport streams. Nevertheless, it has some problems:

- The IEEE 1394 evaluation board has some stability problems, especially in scenarios with many bus resets. It sometimes stops isochronous transmission in case of much asynchronous traffic. Some workarounds have been implemented, in particular a kind of watchdog for isochronous traffic that restarts isochronous transmission if it stops. This evaluation board turned out to be the weakest point in the whole setup.

- The IEEE 1394 evaluation board always packs an integer number of complete MPEG-2 transport stream packets into one isochronous traffic. This is the correct behaviour in case of high bandwidth streams where, on the average, more than one MPEG-2 transport stream packet has to be transmitted per isochronous cycle. This condition is true for streams of an average data rate higher than $188\,\text{Byte} \times 8000\,\text{Hz} = 12.032\,\text{Mbit/s}$. A typical partial MPEG-2 transport stream consisting just of one TV program requires less than half of this bandwidth. Therefore, it should pack half of an MPEG-2 transport stream packet or less into each isochronous packet. Then it would only be necessary to reserve bandwidth for half of an MPEG-2 transport stream packet per isochronous cycle. Instead, this evaluation board always sends a complete packet, when one is available, and several empty packets in-between. The consequence is that bandwidth for a full packet per cycle has to be reserved leading to a waste of isochronous bandwidth. The MPEG-2 transport stream packets should be distributed smoothly over the isochronous IEEE 1394 packets.

- The evaluation board can only transmit data with a data rate of S200 (approximately 200 Mbit/s), it does not support S400 (approximately 400 Mbit/s). If all other devices on the bus support S400, this will also waste bandwidth as transmission takes double of the time that is really necessary. This problem can be fixed by using a newer version of this evaluation board.

- The logic in the FPGA has some deficiencies. In particular, it makes some assumptions that might be violated by some streams [54]. It does not check the CRC. Therefore, in case of transmission errors, incorrect packets might be passed through or incorrect information might be stored. These problems can be fixed by redesigning the FPGA logic.

- As mentioned above, there is this small inconsistency that the source node ID in the isochronous packets is different to the node ID used in control protocols.

[2] There is one more connection from an IO card to the tuner, but this is only used for reset signals.

6.1.2 Software Based Solution

The PC based setup described above requires some quite exotic and, in particular in case of the FPGA card, very expensive additional hardware. The overall system has two interfaces, one to the satellite antenna and one to the IEEE 1394 bus. For both interfaces, low-cost PCI interface cards are available. In case of IEEE 1394, such interface card is also utilised in the setup described above (Subsection 6.1.1), but there it is only used for asynchronous control traffic, the isochronous features are completely ignored. Therefore, the idea of taking a PC with two such interface card and performing all modifications in software seems to suggest itself. Since software based solutions usually offer more flexibility, some deficiencies of the hardware based solution can be avoided, in particular the fact that only complete MPEG-2 transport stream packets are packed into one isochronous packet. But before such solution could be developed, several questions had to be evaluated.

The first question is whether a PC, in particular the PCI bus, is able to handle the involved data rates. The MPEG-2 transport stream has to be transferred from the DVB card to the CPU and, after some manipulations, to the IEEE 1394 card. These transfers are not direct, there is memory in-between. So in fact, the DVB card transfers data via DMA to main memory, the CPU fetches it from there, does some manipulations, and writes it back to memory, then it is transferred via DMA to the IEEE 1394 interface. But since memory uses a separate and faster bus, PCI would be the bottleneck. This means that the data is moved across the PCI bus at least twice. In the worst case of full MPEG-2 transport streams of about 40 Mbit/s, this yields about 100 Mbit/s data transfer over the PCI bus. PCI theoretically offers slightly more than 1 Gbit/s bandwidth, the realistic throughput is somewhere between 800 and 900 Mbit/s. According to this estimation, DVB data would occupy slightly more than 10 % of the available bus bandwidth. This is quite a lot, but it will work, if there are no other applications or devices that occupy much bus bandwidth like high-speed network cards, SCSI cards, etc.

The next question is which kind of DVB card should be used. There are two major kinds of DVB-S cards available. On the one hand, there are cards that basically only contain a DVB tuner and, therefore, provide an unmodified MPEG-2 transport stream. These cards are of rather low costs. On the other hand, there are cards that have additional components on board, in particular an MPEG-2 decoder. These cards are more expensive. Here the decision was taken to use the first version that only features a tuner. An MPEG-2 decoder is not necessary because the stream should be transmitted as an MPEG-2 transport stream on IEEE 1394, and no decoded version is needed inside the system. Furthermore, the gateway should get the transport stream without any modification. In particular, it must be able to detect the original timing of the stream. Therefore, a DVB card that provides an unmodified stream as it was received is the best solution. If hardware already performs tasks like demultiplexing or filtering, it might be more difficult preserve the original timing. So the simpler type of DVB card is the better one for this problem from the technical point of view, and moreover, it enables a low-cost solution.

The most important question is whether it is possible to implement the needed functionality in software, whether drivers and libraries provide the required interfaces. The hardware used is supported by a plenty of different PC operating systems. Open systems have the advantage in comparison to proprietary systems that the source code of all modules is available and, therefore, can be extended or adapted if necessary. Under Linux, the DVB cards in question as well as IEEE 1394 cards are supported. In case of IEEE 1394, two different types of cards are supported, either cards based on link chip TSB12LV21BPGF (PCILynx) by Texas Instruments or

OHCI[3] cards, which are available from many vendors. Since the implementation of isochronous operation is incomplete for PCILynx cards and these cards seem to disappear from the market, it was decided to use an OHCI IEEE 1394 card.

The ideal function principle of the software would be:

1. For each incoming MPEG-2 transport stream packet received from DVB, the IEEE 1394 cycle time at the arrival of the packet has to be determined. The packet is tagged with this time.

2. Filter rules are applied. Packets that should not go through the system are thrown away. Furthermore, modified and new packets like PSI and SI tables are inserted.

3. Isochronous packets are generated. For this purpose, a source packet header is prepended to MPEG-2 transport stream packets, which is determined by the cycle time of arrival of the packet plus a constant offset that takes into account the overall delay of the system. Depending on the data rate of the resulting partial MPEG-2 transport stream, either several MPEG-2 transport stream packets have to be put into one isochronous packet or single MPEG-2 transport stream packets are distributed over several isochronous packets.

4. The isochronous packets have to be scheduled for transmission and to be transmitted at the correct isochronous cycles.

The coupling module between DVB and IEEE 1394 could, in principle, be developed either in kernel space or in user space. The main advantages of a kernel space solution is that in kernel space all functionality of the hardware can be used, whereas in user space only functionality made available by the driver can be utilised. Moreover, in kernel space, there are fewer latency problems, software can directly respond to interrupts, there are no delays caused by communication between user space applications and kernel modules. In kernel space, data can be handed over to IEEE 1394 more directly. Although there are good arguments for a kernel space solution, a user space solution also has some advantages. Software development in user space is much easier, more library functions are available, debugging is much more convenient, the probability of crashing the whole system is much lower leading to a higher stability and reliability. In the Linux kernel developer community, there is a trend to implement only essential functionality inside the kernel and move as much as possible into user space. Also in the Linux IEEE 1394 subsystem, some functionality that had originally been implemented in kernel space has been shifted to user space. Therefore, the decision was taken that the module described here should be implemented in user space if this was possible and the APIs proved to be powerful enough.

The library used on Linux for access to IEEE 1394, `libraw1394`, provides an API that uses mmap, an efficient way to move data between kernel and user space. Using this API, the transmission of isochronous packets can be initiated. Afterwards, the library pulls new isochronous data when it is needed. A callback function, which has to be registered on initialisation, is called. In this function, the user space application can hand over arbitrary data for the isochronous packets. One input parameter of this function is the cycle count at that the packet will be transmitted. The second count is not available, the cycle count wraps around every 8000 cycles. If the application needs information about the second count at which a packet will be

[3] 1394 Open Host Controller Interface Specification is an IEEE 1394 link layer specification jointly developed by many companies [133].

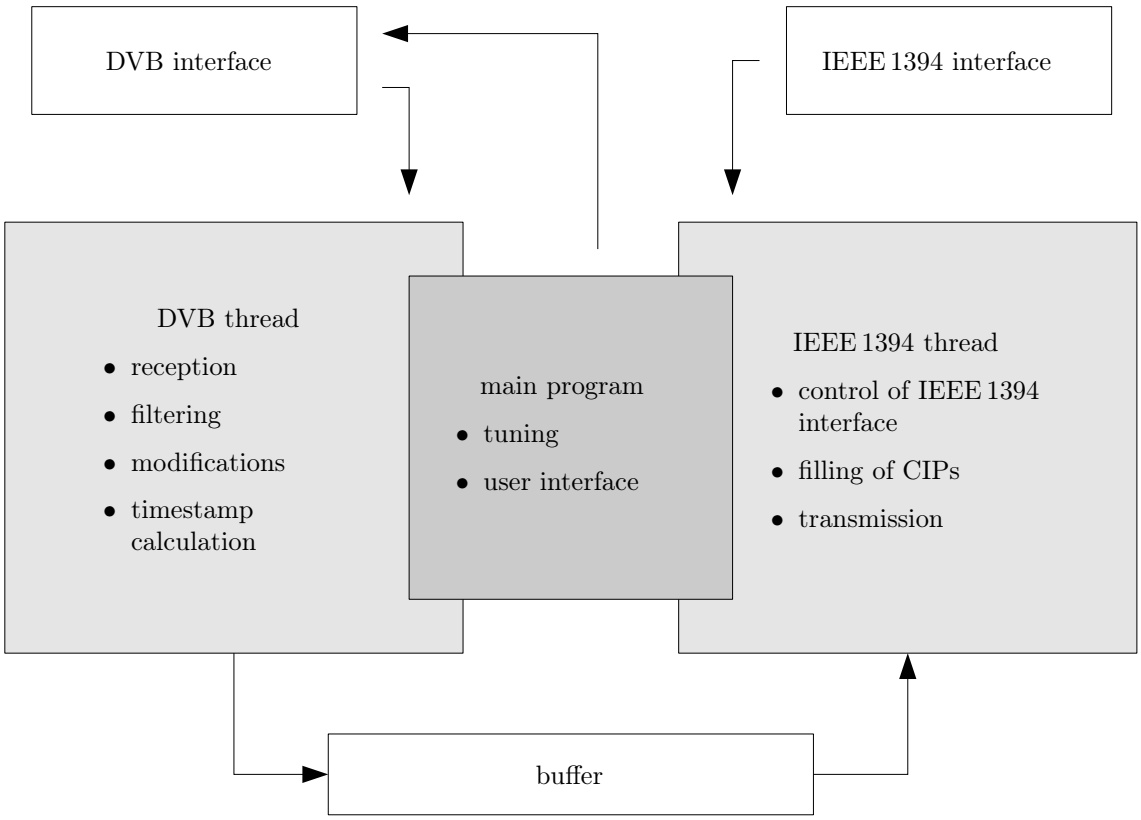

Figure 6.3: Architecture of software based gateway

transmitted, it will have to read the local cycle time register and interpret it correctly, taking into account the packets that are already queued. This functionality is sufficient to precisely queue the isochronous packets for transmission during the correct cycle.

The DVB interface reveals more difficulties for this application than the IEEE 1394 interface. The major problem is that precise timing information about the arrival of the MPEG-2 transport stream packets is not available. It turned out that the driver does not deliver each packet separately, but it always delivers a bunch of packets at once. This means that reading data from the driver will always block for some time, then many packets can be read within a very short time, and then it will block again. Therefore, it is not sufficient to just read the IEEE 1394 cycle time at the moment this data is delivered by the driver. This would yield imprecise arrival times with an unacceptably high variation. As a side note, the same problem would exist in case of a kernel space solution. It is simply not possible to efficiently process each MPEG-2 packet separately, this would require several 10000 interrupts per second. So hardware transfers bigger chunks of data to main memory and then triggers an interrupt. But since a complete transport stream is delivered and its data rate is known to be constant, it is possible to measure the IEEE 1394 cycle time between long chunks of data and divide the difference by the number of MPEG-2 packets in-between. This results in the interarrival time of the MPEG-2 transport stream packets. Using this interval, the arrival time of each single packet can be reconstructed, with a deviation of a constant offset. In order to make this calculation possible, it is very important that the complete transport stream is available at the level at which this calculation is made. If the driver or hardware already filter packets or

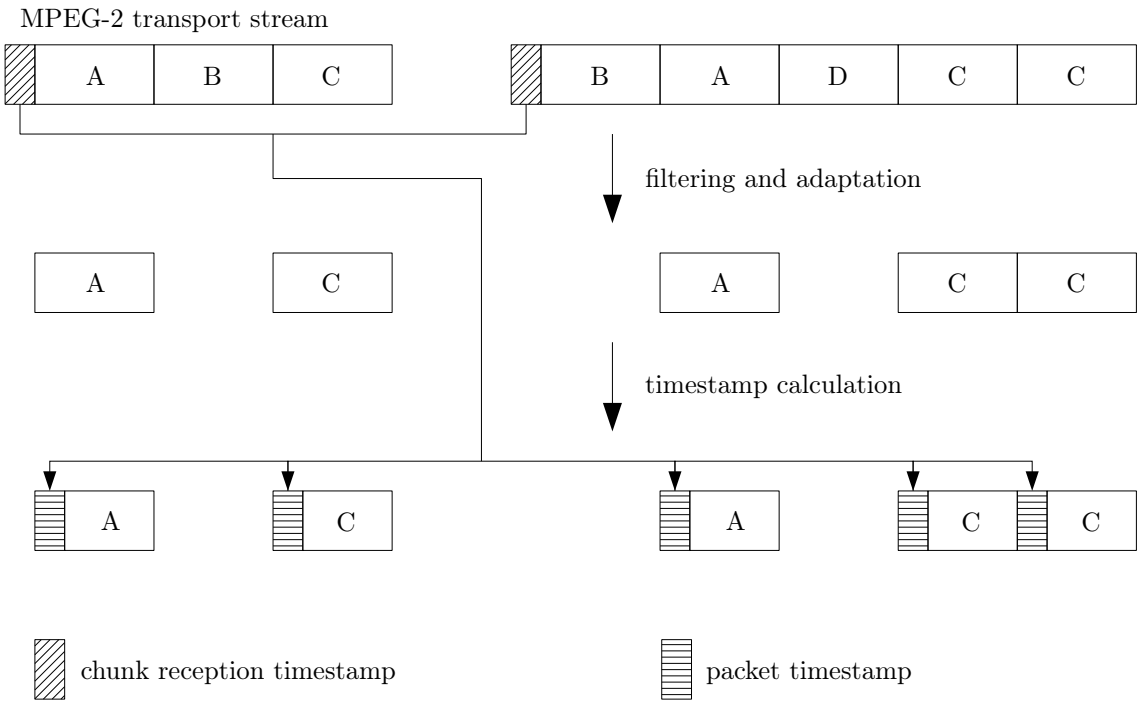

Figure 6.4: Data manipulation and timestamp generation

otherwise modify the number or the order of the packets, this reconstruction algorithm would yield incorrect results.

The actual implementation of the software was done in the course of a diploma thesis [126]. The software architecture is shown in Figure 6.3. It basically consists of an input and an output module, which are running as separate threads and which are coupled by a shared buffer, and a main program, which covers the user interface and has to interact with the input and output modules as well as with the DVB interface for tuning. The modules themselves are modular, they consist of submodules. The program is very flexible: Most parameters, like buffer sizes, timing parameters, etc., can be configured either by startup arguments or at the command line of the text based user interface. The implementation is not complete, it has some limitations, but it proves the concept.

The most interesting part of the software concerns timing, on the one hand the correct generation of timestamps, on the other hand the generation of the isochronous stream on IEEE 1394. The generation of timestamps for the source packet header is demonstrated in Figure 6.4. The DVB API delivers MPEG-2 data in chunks. Consequently, the only reception timestamps that can really be measured are the handover timestamps of these whole chunks, not the reception time of single packets. In experiments, the size of the chunks was between 325 and 351 packets, the average chunk size was 341.3 packets or 64170.7 bytes. Depending on the data rate of the involved transponder, this typically corresponds to an interval of 13 ms or 15 ms between these chunks. Data is really delivered in very regular intervals. Of course, since no real-time operating system is used, this time can vary a lot, especially in case of high process load, but at low loads, the deviation from this regular interval is usually below 3 ms. After each received chunk, the program reads the current IEEE 1394 cycle time. This cycle

time is regarded as the time at which the first MPEG-2 packet of the chunk was received. This decision was probably not the best one. It would be better to regard it as the reception time of the last packet in the chunk or of the first packet of the next chunk because in case of bigger variations of the chunk length, it could rather be expected that the time when the buffer is read from the DVB subsystem more precisely corresponds to the reception times of these packets. However, as long as the lengths of the chunks are similar, it does not make a big difference. For all other packets in the chunk, the reception times have to be interpolated. In the actual implementation in the demonstration setup, the chunk reception timestamps are really taken—without adaptation—as packet timestamps for the first packets, which are used for calculation of the timestamps in the source packet header. This is a clear design flaw because the measurement errors of the chunk reception timestamps are directly transferred to the packet timestamps of these first packets and, in consequence, also to all other packets. For a proper solution, the packet timestamps of all packets, even the first ones of the chunks, have to be calculated using an algorithm as described below. The chunk reception timestamps are used as base for calculating the interval between consecutive packet timestamps, but they should not be used directly as packet timestamps for some packets. Since the packets are sent at a constant data rate, the interval between two packets can be calculated as:

$$x(n) = \frac{t(n) - t(n-1)}{c(n)} \tag{6.1}$$

In this formula $t(n)$ is the time the current chunk was read and $c(n)$ is the number of MPEG-2 transport stream packets within this chunk. The calculation is performed as number of isochronous cycles. There are always measurement errors. The major problem is that it is not possible to measure the IEEE 1394 cycle time exactly at the same time the DVB data is read from the DVB system. It can either be done immediately before or after reading DVB data. Between these two actions, it is always possible that the scheduler schedules some other processes and, therefore, causes a delay, which is typically in the range of milliseconds or above on Linux on the i386 platform. It is possible to measure the maximum error by sampling the IEEE 1394 cycle time before and after reading DVB data. In this case, the difference between these two values would be an upper limit of the error. A second problem is that there are latencies within the DVB subsystem. When data is read from this subsystem, it might have resided there for some time. In order to overcome these errors, an averaging mechanism is employed. It should not calculate the average from start of the program because this would not work if the data rate drifted slightly over time, and it would not heal temporary error conditions like packet loss. On the other hand, averaging must take into account a longer time in order to achieve the required accuracy. It was decided to use a floating averaging with exponentially decreasing weights for older measurements. This allows a very simple and efficient calculation, where the latest sample has the highest weight, but also older ones have a significant impact. The average interval between two packets is calculated as:

$$\bar{x}(n) = w \cdot x(n) + (1 - w) \cdot \bar{x}(n-1) \tag{6.2}$$

Here w is the weight of the currently measured interval. The initial value $\bar{x}(0)$ is assumed 0.3 cycles per packet. This is approximately the range of values that are to be expected on the typical transponders of the Astra satellite system, which either use a symbol rate of 27.5 MBd with an inner FEC (forward error correction) of a code rate 3/4 or a symbol rate of 22 MBd

and an inner FEC of code rate 5/6. This leads to the following theoretical packet intervals:

$$x = \frac{1\,\mathrm{s}}{27500000\,\mathrm{symbol}} \cdot \frac{1\,\mathrm{symbol}}{2\,\mathrm{bit}} \cdot \frac{4}{3} \cdot \frac{8\,\mathrm{bit}}{\mathrm{Byte}} \cdot \frac{204\,\mathrm{Byte}}{\mathrm{packet}} \cdot \frac{8000\,\mathrm{cycle}}{1\,\mathrm{s}} = 0.317\,\frac{\mathrm{cycle}}{\mathrm{packet}} \tag{6.3}$$

and

$$x = \frac{1\,\mathrm{s}}{22000000\,\mathrm{symbol}} \cdot \frac{1\,\mathrm{symbol}}{2\,\mathrm{bit}} \cdot \frac{6}{5} \cdot \frac{8\,\mathrm{bit}}{\mathrm{Byte}} \cdot \frac{204\,\mathrm{Byte}}{\mathrm{packet}} \cdot \frac{8000\,\mathrm{cycle}}{1\,\mathrm{s}} = 0.356\,\frac{\mathrm{cycle}}{\mathrm{packet}} \tag{6.4}$$

Each packet requires 204 bytes instead of 188 bytes because of 16 bytes outer FEC. Since all the parameters required to calculate the theoretical packet interval are known in advance, it is a trivial improvement to the program to calculate the theoretical packet interval as initial value $\bar{x}(0)$. As it will be pointed out, this improvement is required if the tight timing requirements of MPEG-2 have to be met.

It should be noted that as an alternative to this exponentially weighted average it would, of course, also be possible to calculate the average over a floating window with the same weight for all $x(n)$. This is obviously equivalent to the average packet reception rate over several chunks. The main argument for the chosen algorithm was that implementation is much easier using the recursive formula (6.2). Only one value has to be stored, it is not necessary to store and maintain a long list of old reception timestamps and chunk sizes.

An interesting decision is which weight w to use for calculation of the floating average. If it is chosen too low, the calculation will converge too slowly, and it will not follow the current data rate. If it is too high (e. g. between 0.5 and 1), in fact, only the latest measured $x(n)$ will influence $\bar{x}(n)$ significantly, and therefore, inaccuracies will not get attenuated sufficiently. The standard specifies some timing constraints for MPEG-2 transport streams [49]. Since the inaccuracies and jitters of the original stream are unknown, it is also unknown which additional errors might be added by this gateway system to still meet the requirements of the standard. The following considerations assume that the original stream provides perfectly correct timing and the gateway could, therefore, introduce additional timing errors up to the maximum allowed by the standard. A tolerance for PCRs is specified as ±500 ns [49]. For actual applications, the requirements might be a bit more relaxed, but to be compliant to the standards, the timestamps in the source packet header should have this accuracy of ±500 ns. In fact, this limits the jitter of the overall delay through the gateway to ±500 ns. This is a constraint that can be met in a hardware based solution as shown in Subsection 6.1.1, but not in the software based solution discussed here. The arrival time is not known at that exact level, the system just measures the interarrival interval, the absolute variation of delay can accumulate significantly over time. Since the calculated $\bar{x}(n)$ is used for generation of timestamps for long chunks of data, errors of $\bar{x}(n)$ do not lead to a real jitter of the transport delay, but rather to a systematic drift. Therefore, the error introduced by the gateway system should not be seen as inaccuracies of single PCRs, but rather as a slight variation of the bit rate and, since the values of the PCRs and their positions within the bit stream are not modified, a variation of the system clock frequency. The allowed tolerance for the system clock frequencies is 30 ppm. This leads to a tolerance of the time between PCRs and, therefore, of the time between packets of also about 30 ppm.

Based on the numbers given above, it is assumed that the maximum error of Δt is 3 ms and the minimum chunk size $c(n)$ is 325 packets. These are the numbers obtained by practical tests with the DVB API. This leads to a maximum error of $x(n)$:

$$e = \frac{\max \mathrm{error}(\Delta t)}{\min(c(n))} = \frac{3\,\mathrm{ms}}{325} = 9.23\,\mu\mathrm{s} = 0.0738\,\mathrm{cycle} \tag{6.5}$$

Now it is proposed to choose the weight w that a single maximum error changes $\bar{x}$ from a previously correct value $\bar{x}(n-1)$ to $\bar{x}(n) = \bar{x}(n-1)(1+p)$, where $p = 30\,\text{ppm}$:

$$\bar{x} \cdot (1+p) = w(\bar{x}+e) + (1-w)\bar{x} \tag{6.6}$$

$$w = \frac{p\bar{x}}{e} = \bar{x} \cdot \frac{30\,\text{ppm}}{9.23\,\mu\text{s}} = \bar{x} \cdot \frac{30\,\text{ppm}}{0.0738\,\text{cycle}} = \bar{x} \cdot 3.25\,\text{s}^{-1} = \bar{x} \cdot 0.000406\,\text{cycle}^{-1} \tag{6.7}$$

For the theoretical packet intervals given in (6.3) and (6.4), this leads to $w = 0.000128$ and $w = 0.000144$[4]. There is no danger that successive errors of $t(n)$ could accumulate leading to too high error of $\bar{x}(n)$ for these reasons:

1. If the error of $x(n)$ was maximum, it would mean that the error of $t(n)$ was at the maximum. If the error of $t(n+1)$ is again at the maximum, $x(n+1)$ will be correct, and therefore, $\bar{x}(n+1)$ will move towards the correct value. In all other cases the error of $x(n+1)$ will have the opposite sign of the error of $x(n)$, and hence, both errors partly compensate each other.

2. If the error of $x(n)$ was not at the maximum, the error of $t(n)$ could have been below its maximum. Therefore, it is possible that the error of $t(n+1)$ is bigger than the error of $t(n)$ leading to an error of $x(n+1)$ that has the same sign as the error of $x(n)$ and moving $\bar{x}(n)$ into the same direction. But, as it can be seen easily, if a modification of $\bar{x}(n)$ is performed in several small steps caused by small errors of $x(n)$ in the same direction, the influence will be always less than in case of one big step caused by an error of $x(n)$ that equals the sum of the small errors because, in case of one step, the complete error is weighted with w, whereas in case of several steps the older errors have less influence.

The weight w chosen according to these considerations is very small. This has the positive effect that the influence of single measurement errors is very low, which, in fact, was the criterion for calculating w this way. But it has the negative effect that convergence is very slow, incorrect values of $\bar{x}(n)$ need very much time to move towards the correct value. Simulations have shown that it would take about two minutes to get from an error of 100 ppm inside the tolerance of 30 ppm and one more minute to permanently stay within this 30 ppm margin. This is an interesting number because 100 ppm is the tolerance of the IEEE 1394 cycle time [19]. Therefore, it can be expected that the initial value $\bar{x}(0)$ could have approximately this error if the theoretical value was chosen, as proposed above. For bigger errors, it takes significantly more time to converge to the correct value. E. g. if the initial error was 50 %, it would take more than 16 minutes to get into the tolerance margin.

In the demonstration project, much bigger weights w, up to 0.2, were used [126]. There was no special reason for such big weights. The timing constraints were not investigated and a static $\bar{x}(0)$ was used that caused a huge initial error. This made a big weight w necessary in order to achieve reasonable convergence times. The big errors of the timestamps caused by the big w were no problem in the demonstration setup because the client application was very tolerant to timing violations. However, as it was pointed out here, the proper solution is to use smaller weights w and more accurate estimations for the initial value $\bar{x}(0)$.

Simulations for weights w chosen according to (6.7) have, furthermore, shown that, once the correct value has been reached, the error stays significantly below 30 ppm, it just goes up to about the half, 15 ppm. The reason is that the assumptions in (6.7) are a bit too pessimistic.

[4] Both values were rounded down in order to keep the error below the tolerance.

In fact, the error e of each iteration is not independent from the previous iteration. If there was a big error in one direction, the next error has to be small or to have the other sign. If $\bar{x}(n)$ as well as $x(n)$ have no error (this could be regarded as the steady state), the next error of $x(n+1)$ can be at most half of the maximum error, leading to an error of $\bar{x}(n+1)$ of half of the allowed tolerance, so $\bar{x}(n+1)$ would have an error of 15 ppm. Now, $x(n+2)$ could have the maximum error, but in the opposite direction. Therefore, it would move $\bar{x}(n+2)$ 30 ppm into the right direction and over the correct value, leading again to an error of 15 ppm in the other direction. The only way get higher error values of $\bar{x}(n)$ is to have very many steps into one direction—gradually moving towards a maximum error of $t(n)$ while keeping the effect on $\bar{x}(n)$ low—and then one maximum step into the other direction. The smaller the value of w, the more small steps are required, making it very unlikely to happen. This means that it would be acceptable to increase w almost by a factor 2.

There is also a maximum tolerance of the rate of change of the system clock frequency, which should be within 75 mHz/s or 10 ppm/h [49]. It can easily be seen that this condition cannot be fulfilled when the system is recovering from an error like an inaccurate initial configuration. As shown above, it could drift about 30 ppm per minute if the initial value was not too far away from the final value, in case of bigger errors the change rate is much higher. Once the final value is reached, the variations of $\bar{x}(n)$ and their influence to PCR timing might be interpreted as PCR jitter rather than systematic drift. This view is supported by the fact that, for one given service within the transport stream, not every chunk of data received from the DVB API contains a packet with a PCR. MPEG-2 specifies that PCRs shall be transmitted at least every 100 ms, DVB has the same requirement [49, 134]. This means that the adaptations of $\bar{x}(n)$ usually happen at a higher rate than transmissions of PCRs. Therefore, errors of $\bar{x}(n)$ do not cause a systematic drift over several PCRs. It is assumed that the accumulated error over a complete chunk should be less than the 500 ns specified by MPEG-2 as maximum jitter of PCRs [49]. Since the maximum chunk size is 351 packets during normal operation the maximum change between two $\bar{x}(n)$ should be $500\,\text{ns}/351 = 1.42\,\text{ns}$. A slight modification of (6.7) leads to the required weight w to fulfil this criterion:

$$w \leq \frac{1.42\,\text{ns}}{e} = \frac{1.42\,\text{ns}}{9.23\,\mu\text{s}} = 0.000154 \tag{6.8}$$

Hence, it can be seen that this jitter requirement is fulfilled for the weights calculated above according to (6.7) once the algorithm has converged near to the final, correct value of $\bar{x}(n)$. It is, furthermore, obvious that the demonstration setup with weights up to 0.2 cannot fulfil these requirements. If w was chosen so that the clock drift requirement was also fulfilled during startup or recovery from errors, it would obviously be necessary to select w so that convergence from an error of 100 ppm would take at least 10 hours. In order to limit the drift rate even between two consecutive calculations of $\bar{x}(n)$, this would lead to convergence times in the range of years. For calculation of these values, the drift rate of 10 ppm/h has to be multiplied with the duration of the chunks (about 12.88 ms and 14.46 ms for the shortest chunks in case of two considered types of transponder) leading to maximum allowed drifts of $3.58 \cdot 10^{-5}$ ppm/chunk and $4.02 \cdot 10^{-5}$ ppm/chunk. With slight adaptations to (6.7), w can easily be calculated to be in the range between about $1.5 \cdot 10^{-10}$ and $2.0 \cdot 10^{-10}$, depending on the exact stream parameters. It can be easily verified in simulations that it would need years to converge. Hence, for practical purposes, the algorithm would not converge at all. It should be noted that IEEE 1394-1995 does not specify a maximum drift rate for its cycle time. Since the drift rate of the reference clock has no specified limit, it is not possible to limit the drift rate of a clock derived from it,

anyway. This fact, moreover, indicates that too slow algorithms have the problem that they cannot follow drifts of the reference clock—the IEEE 1394 cycle time in this case—sufficiently fast.

If the system transmits data at a rate too slow for some time, data will accumulate in its internal buffer. If it transmits too fast, the buffer will be drained. The calculated packet timestamps directly determine the transmission process, which will be explained in detail later in this subsection. Therefore, longer intervals between the packet timestamps, i. e. bigger $\bar{x}(n)$, lead to a slower transmission on IEEE 1394, shorter intervals lead to a faster transmission. The effect of the long convergence times on buffer usage is an important criterion whether this configuration is usable. As simulations have shown, the effect on the buffer usage in case of starting at an error of $\bar{x}(0)$ of 100 ppm is about $\pm$50000 bytes. During normal operation within the 30 ppm margin, the variation of buffer usage is significantly lower, in simulations around $\pm$7000 bytes. These numbers are valid for the higher data rate streams according to (6.3) and the parameters calculated above. For the lower data rate streams according to (6.4), the effect on buffer usage is a bit lower. These numbers are consistent with the fact that 100 ppm of the higher data rate stream is equivalent to 515.625 Byte/s, leading an upper limit of about 93000 bytes in three minutes if there were no modification towards the correct data rate. For bigger initial errors, the buffer usage is, of course, significantly higher; in case of around 50 %, it reaches several megabytes. The conclusion is that the system has to buffer at least around 50000 bytes corresponding to 266 MPEG-2 transport stream packets. The transport delay of the system is, thus, at least 85 isochronous cycles or 10.5 ms. This is less than the typical chunk size received from the DVB API. Therefore, the gateway could commence transmission immediately after receiving the second chunk of data (in fact, slightly before), the probability for running out of data would be low. The limiting factor for the buffer size is, in this scenario, the bursty behaviour of the DVB API rather the smoothing process on the IEEE 1394 side. These simulations assume that, after each received chunk, exactly the number of packets is taken out of the buffer for transmission that have a packet timestamp smaller than the reception time of the currently received chunk. In the real system, the IEEE 1394 subsystem takes bigger chunks of data into an internal buffer for transmission. This leads to an additional variation of the buffer usage inside the application. In order to make these considerations about the buffer usage also apply to the real system, it is necessary that calculation of the current buffer usage not only takes into account the buffer of the application between the DVB input module and the IEEE 1394 output module, but also the data waiting in the buffer of the IEEE 1394 subsystem for transmission. Since the current IEEE 1394 cycle time as well as the packet timestamp of the last packet in the buffer of the IEEE 1394 subsystem are known, the amount of data inside this buffer can be estimated easily.

The buffer requirements were calculated for full transport streams. The gateway modifies the transport stream and transmits a partial transport stream (see Figure 6.4). Therefore, the numbers above that refer to memory could usually be reduced for partial transport streams. The numbers related to time units stay the same because the partial transport stream is essentially generated by cutting off data from the full transport stream, all timing properties stay untouched. However, the system is designed for full transport streams, which are the "worst case" of partial streams. It will, consequently, support all kinds of partial transport stream.

The algorithm described above allows implementation of the system in a way that most timing requirements of the MPEG-2 standard [49] can be fulfilled except the clock drift in some situations, in particular system startup. It should be mentioned that the real timing requirements depend very much on the application. For example, the timing requirements will

be very tough if the system clock is used to synthesise video signals. If the target device is a PC with a software MPEG-2 decoder, the timing requirements will be much more relaxed. Therefore, in many applications, the weight w can be chosen some orders of magnitude greater than calculated above. This will result in faster convergence and recovery from error situation and less buffer requirements. Furthermore, it will mitigate the requirement to exactly calculate the initial value $\bar{x}(0)$. If the algorithm converges sufficiently fast, initial errors will be acceptable. The demonstration setup that was really implemented had very light timing requirements, and this system has defined many parameters different to the considerations above [126].

Although this algorithm leads to a transmission data rate on IEEE 1394 that is equivalent to the reception rate on the DVB side, it does not guarantee a certain buffer usage. The buffer usage that is reached depends on the initial error of $\bar{x}(0)$. But due to inaccuracies, a drift of the buffer usage has to be expected even in the steady state. Therefore, an additional mechanism is necessary that keeps the buffer usage within some defined range. Such mechanism is not necessary in the demonstration setup [126] because there each chunk is "resynchronised" to the measured reception time and, therefore, cannot drift away. Since this leads to very inaccurate timestamps, violates all the above-mentioned timing requirements and also leads to non-monotony of the timestamps in the source packet headers, this mechanism should be avoided and is not regarded a a possible option here. In fact, it makes the smoothing mechanism described above completely useless. Hence, it is proposed to use the smoothing algorithm described above as long as the buffer usage stays within a certain range. This means, after each received chunk, the new $\bar{x}(n)$ is calculated, the timestamp in the source packet header of the first packet of the chunk corresponds to the timestamp of the last packet of the previous chunk plus the current $\bar{x}(n)$. The difference between the timestamps of the packets within the chunk is always $\bar{x}(n)$. The packets are put into a buffer where they will be taken from for transmission on IEEE 1394. This buffer has a target buffer usage and a range around marking a target buffer usage region. At the first glance, it seems obvious that this range should be greater than the chunk size of the DVB API. Otherwise, a single new chunk would always move the buffer usage outside this target range if it was inside before reception of this chunk. This argument is not completely true because the buffer usage is always examined before the new packets are added to the buffer[5]. Therefore, it would not be detected that the buffer usage leaves the target buffer usage if this only happens always for a short time after adding the new packets and gets healed until the next arrive. The overall size of the complete buffer must, of course, be bigger than the biggest chunk of DVB data, but the target range could be smaller. Consequently, the typical chunk size only gives a rough suggestion for the size of the target buffer range. The real requirement is that it must be bigger than the amount of incoming data equivalent to the jitter of reception process from the DVB API. And it should not cause an unacceptably long delay. It is suggested to use 2048 MPEG-2 transport stream packets, which is almost 6 times the maximum typical chunk size, corresponds to almost 30 times the jitter of the arrival of chunks of DVB data and to about 90 ms maximum delay. All these numbers, as well as the numbers in the following paragraphs, are calculated based on full transport streams. In order to keep the timing properties, like the transfer delay, the same for partial transport streams while keeping the same buffer parameters, in particular the target buffer usage region, it is important to perform the calculation of the current buffer usage based on the full transport stream and also count the packets that are not part of the partial transport stream. One solution to this problem

[5] It could also be examined always after adding the new packets. The argumentation would stay almost the same.

is to postpone the filtering to a later step, maybe perform it immediately before transmission on IEEE 1394. If filtering is done immediately when inserting the packets into the buffer, the system will have to keep track also of the removed packets. It can either put some placeholder packets into the buffer—that must be discarded in the IEEE 1394 output module—, or it must store the number of removed packets between retained packets and maintain a buffer usage count that is modified appropriately when packets are taken from or added to the buffer.

If the buffer usage drifts outside this range, the system should slightly increase or decrease the transfer rate in order to move the buffer usage into the desired range. One possibility is to use $\bar{x}(n)$ together with a modifying factor that is proportional to the deviation of the buffer usage from the corresponding target buffer usage range border. The modifying factor should be chosen small so that in normal operation the additional error should be below 30 ppm. In fact, the sum of both errors, the error caused by calculation of $\bar{x}(n)$ and the error introduced by this modifying factor, should be below 30 ppm. As already discussed, the weights w that were designed for a maximum error of 30 ppm in worst case scenarios usually lead to errors below 15 ppm because of the very little probability of the worst case. This means that, if w was chosen as calculated above, there would be a remaining margin of about 15 ppm. One possibility to calculate this factor is:

$$f(u) = \begin{cases} 1 - C \cdot (u + 1024) & \dots \quad \text{for } u \leq -1024 \\ 1 & \dots \quad \text{for } -1024 < u < 1024 \\ 1 - C \cdot (u - 1024) & \dots \quad \text{for } u \geq 1024 \end{cases} \tag{6.9}$$

Here $f(u)$ is the modifying factor, u is the deviation of the buffer usage from the target buffer usage, which is a half-full buffer, and C is a constant chosen suitably.

As simulations have shown, the variations of the buffer usage during normal operation is typically below ± 40 packets. Therefore, it can be expected that the buffer usage usually does not leave the desired range more than 40 packets.

Based on the numbers given above it is proposed to add buffers of 1024 packets on both sides of the target range, leading altogether to 4096 packets buffer, which is equivalent to 770048 bytes. This, furthermore, corresponds to a delay between 150 ms and 200 ms, depending on the exact data rate. The target buffer usage is half of the buffer. Therefore, the typical transport delay is less than 100 ms, which is perfectly acceptable. A reasonable proposal for C is 0.2 ppm/packet. This would result in a maximum error of about 200 ppm at the limits of the buffer and less than 10 ppm during normal operation. A graph of this function is shown in Figure 6.5. In fact, C could be even chosen an order of magnitude bigger. During normal operation the average buffer usage should not move that near to the border of the target area because once it comes that near that some chunks move the buffer usage slightly outside the buffer area, this correction mechanism will ensure to move it back towards the target buffer usage. On startup, the buffer should be filled up to the target buffer usage, before transmission is started, in order to avoid filling up the buffer by too slow transmission with too big errors.

There are many variations possible of the mechanism described above, and many parameters could be modified. It is possible to substitute the piecewise linear function $f(u)$ in (6.9) with a function that is more flat at the borders of the target area and has a higher slope when a buffer overrun or underrun is near. Another variation is to introduce some kind of hysteresis in case a modification of the transfer rate is necessary. So it would be possible to modify the rate not only up to the point where the buffer usage is within the valid area again, but to bring the buffer usage again back to the middle of the buffer. Two parameters that can be

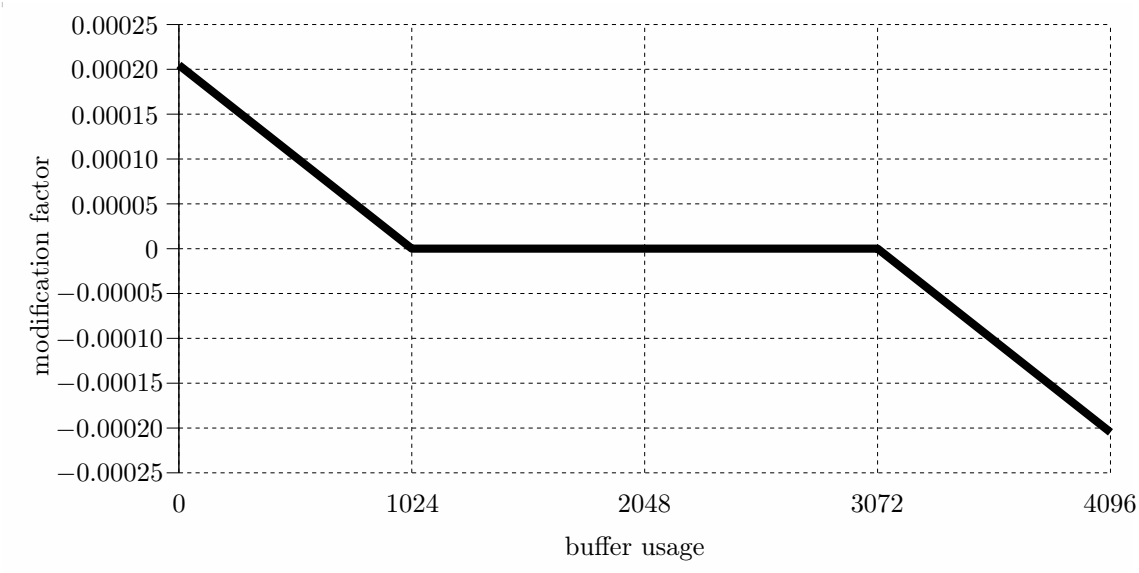

Figure 6.5: Modification factor $f(u)$ as function of the buffer usage

adapted are obviously the various buffer sizes, both the size of the target area and the areas in the margin where modifications of the transfer rate are performed, and the constant C in (6.9). It should be kept in mind that bigger buffers are equivalent to a higher transfer delay and, therefore, also for a longer startup process. On the other hand, bigger buffers will be necessary, if there are higher fluctuations in the incoming data or if the chunks are bigger. So the timing behaviour of the system, which might also depend on configuration parameters of the scheduler of the operating system, significantly influence the necessary buffer sizes. This also means that, if the software is ported to another system, adaptations of these buffer sizes might be necessary. Variations of the constant C are an interesting possibility to modify the behaviour of the system. If small values are used, the errors introduced by this mechanism will obviously be smaller. On the other hand, the variations of the data rate that can be handled will be smaller and it will take more time to come back into the target area of the buffer. This leads to the requirement for bigger buffers. If big values are chosen for C, the buffer usage can be corrected quickly and variations of the incoming data rate can be compensated better at the cost of higher possible errors.

One interesting variation is to use only this mechanism of modifying the transmission rate based on the buffer usage for synchronisation and abandon the mechanism for calculating the floating average of the arrival data rate based on (6.2). This option is evaluated more in detail now. It turns out that this approach is also a possible solution. It is obvious that the steady state buffer usage depends on the incoming data rate, and it is not possible to fix some target buffer usage exactly, unless the incoming data rate is exactly what the system expects based on the IEEE 1394 cycle time. Therefore, it does not make sense to introduce some flat area where $f(b)$ is exactly 1 around a target buffer usage because buffer usage will usually not stay within this area as the incoming data rate will always at least slightly differ from the expected one. Hence, $f(u)$ is a linear function:

$$f(u) = 1 - C \cdot u \tag{6.10}$$

Here u is the deviation of the buffer usage from half of the buffer. The deviation from the

target buffer usage that will be reached can be calculated easily as:

$$u = \frac{T_{\text{ref}} - T_{\text{act}}}{C \cdot T_{\text{ref}}} \tag{6.11}$$

T_{act} is the actual average interval between the packets of the incoming data stream. T_{ref} is the reference interval, the ideal interval that is expected by the system and that would cause the buffer to stay filled up to the half (the target buffer usage).

In this scenario, two parameters can be adjusted, the buffer size and the constant C. The major properties that are affected by these parameters are the range of supported deviations from the reference data rate, the maximum timing errors at the output, and the transfer delay. It is obvious that the biggest disadvantage of this approach is that it is necessary to know the incoming data rate very accurately. While the algorithm for calculating the floating average of the arrival rate in (6.2) enables adaptations to a greater range of data rates from bigger initial errors depending on the weight w, this mechanism only allows small deviations. For a given buffer size and C, the range of supported data rates can be calculated easily. Or the other way round, for a given C and a range of supported input data rates, the necessary buffer size can be calculated. For the parameter C, the most important criteria are the convergence speed and the error caused by variations of buffer usage. As simulations have shown, the buffer usage at the time where the chunks are received jitters slightly below ± 40 packets. In order to stay within the 30 ppm margin around the actual incoming data rate, C has to be chosen at most 0.75 ppm/packet. In order to deal with errors of 100 ppm of the incoming data rate (or of the reference clock, depending on the point of view), it would be necessary to use at a buffer of least around 350 packets. Of course, the buffer should be much bigger in order to get some safety margin. Moreover, it shows that also in this scenario the limiting factor for the minimum buffer size is the DVB API, not the transmission process. Convergence is even faster than in case of the algorithm where the arrival rate is measured. However, if it was taken into account that the weight w as calculated above according to (6.7) usually yields errors less than 15 ppm, the convergence speeds would be in similar range if the parameters were adjusted for the same typical errors.

The conclusion is that, if the incoming data rate is known, it will not be necessary to employ a mechanism to actually measure the timing of data reception. This makes the algorithm less complicated and also leads to good results. But this approach also has some drawbacks. Temporary bigger variations of the incoming data stream have direct influence to the outgoing stream, they are not attenuated as in case of using the averaging mechanism.

The length of the elaboration above clearly emphasises that the generation of correct timestamps is the central problem of the software based solution. But other tasks have to performed as well. As already shown in Figure 6.4 important features are the filtering and the manipulations that transform the full MPEG-2 transport stream into a partial transport stream. The filtering is based on the PIDs (packet identifiers). It is important that the filtering is not performed inside the DVB API because the gateway requires the exact position of each packet within the transport stream. The assumption that the time between two successive packets is constant, which the complete timestamp generation mechanism relies on, is only valid for full transport streams. In the partial transport stream, where some packets have been cut off, this is no more fulfilled. Therefore, the gateway either has to generate the timestamps before the stream is modified, or it has at least to store the exact positions of the packets within the full transport stream so that later on, after filtering, it is possible to reconstruct the reception time. The latter mechanism is used in the demonstration setup, as it can be seen in Figure 6.4.

Although filtering is based on PIDs, the system allows specifying program numbers and automatically sets the correct PID filters. In addition to filtering, it is also necessary to modify PSI (program specific information) tables. The system replaces the PAT (program association table) with a modified version that only contains the programs that are actually part of the partial transport stream. Furthermore, the PMT (program map table) would have to be modified, if not all components of a program were to be transmitted, and DIT (discontinuity information table) and SIT (selection information table) have to be inserted. Since the implemented system is just a prove of concept setup, it does not offer these features, but it is clear, that they could easily be added.

The last manipulation of data is the generation of the isochronous packets on IEEE 1394 according to IEC 61883-4 [63]. This standard puts some constraints on generation of the packets. One isochronous packet might either contain one, two, or four data blocks (corresponding to eighths, quarters, and halves of MPEG-2 transport stream packets) or a multiple of eight data blocks (corresponding to an integer number of complete transport stream packets). Therefore, in order to be compliant to the standard, the gateway must not arbitrarily put data blocks into isochronous packets in order to make the stream as smooth as possible. Instead, for low data rates requiring less than one transport stream packet per isochronous packets on average, the gateway has to send either an empty packet or a packet containing a certain fix number of data blocks during each isochronous cycle. For higher data rates, requiring more than one transport stream packet per isochronous cycle on average, the gateway switches between two numbers of complete transport stream packets per isochronous cycle.

The gateway in the demonstration setup takes a fixed number of transport stream packets from the ring buffer, where they have been put after stream modification and timestamp generation. This number of packets is called "packet smooth interval" [126]. From the timestamps of the first and the last packet, it calculates the average number of transport stream packets per isochronous packet. Based on this number, it determines how many data block are to be put into one isochronous packet. Then it uses an algorithm, which is described below, to distribute the full packets smoothly over the respective number of isochronous cycles. The timestamp in the source packet header corresponds to the reception time in the gateway, determined as described above in detail, plus some constant offset. The offset has to be chosen that on the receiving nodes the timestamp in source packet header points into the future, i. e. refers to a time that is slightly after the reception time there. This makes clear that this constant has to be selected coordinated with the packet smooth interval because the bigger the smooth interval the bigger is also the maximum displacement within the output stream. This displacement is caused by unsteadiness of the packets that pass the transport stream modification. It can be seen easily that an upper limit for the displacement is designated by the difference between the timestamps of the last and the first packet under consideration. The biggest weakness is that since the algorithm takes a fix number of transport stream packets, the interval in units of time is not known in advance, and it will usually vary for each chunk of data. Therefore, it is not possible to select the packet smooth interval and the constant offset for source packet header generation together in a way to ensure that there will be no late packets at the receiver. The solution that is proposed here is not to perform the generation of isochronous packets for a fixed number of transport stream packets, but rather for a fixed number of isochronous cycles.

The system has to use some constant number of isochronous cycles that packets are to be created for at once. To make the algorithm simpler, this number should be a multiple or a of eight. It is also possible to take 4, 2, or 1 cycles, but in this case, it would not be possible to put eighths, quarters, or halves of MPEG-2 transport stream packets into single isochronous

Algorithm 6.1 Distribution of data blocks over isochronous packets

$accumulator \Leftarrow 0$
determine size and number of entities to distribute over isochronous cycles
$ratio \Leftarrow \frac{\text{total number of entities}}{\text{number of isochronous packets}}$
while not all isochronous packets filled **do**
  $accumulator \Leftarrow accumulator + ratio$
  send packet containing $\lfloor accumulator \rfloor$ entities
  $accumulator \Leftarrow accumulator - \lfloor accumulator \rfloor$
end while

packets. To illustrate the mechanism, it is now outlined for 16 cycles. When the isochronous packets for the cycles T to $T + 15$ are to be generated, it considers the packets in the buffer that have a reception timestamp from the interval $[T - T_0, T + 16 - T_0[$, where T_0 designates the time (cycle) a packet received at the beginning of cycle 0 (reconstructed reception time) should be sent or, in other words, the delay of packets within the gateway. This constant depends on the startup process, when isochronous transmission is started. If there are two or fewer than two packets in the buffer with a timestamp from this interval, the packets will be split into 8 data blocks and distributed over the available isochronous packets, so each isochronous packet will either be empty or contain one data block. If there are four or fewer than four, but more than two packets in the buffer with a timestamp from this interval, the isochronous packets will be either filled with two data blocks (i. e. quarters of transport stream packets) or stay empty. In case of up to eight packets within this interval, isochronous packets will be filled with halves of transport stream packets (four data blocks). If there are up to 16 packets, each isochronous packet that will not stay empty will contain eight data blocks corresponding to exactly one isochronous packet. In case of more packets, each isochronous packet has to be filled with an integer number of transport stream packets or, in other words, a multiple of eight data blocks; all packets will contain some minimum number of transport stream packets, some will contain one packet more. One strategy how to distribute the full isochronous packets or, in general, the packets of the maximum size over the available cycles is shown in Algorithm 6.1. This algorithm first has to find out the size of the entities to put into isochronous packets (eighths, quarters, halves or complete MPEG-2 transport stream packets) as described above. It calculates ratio between the total number of such entities and the number of isochronous packets. Then for each isochronous packet, this ratio is accumulated in an accumulator. If there are enough fractions accumulated, the current packet is filled with the biggest integer number of entities contained in the accumulator, which is then decreased by this number. Table 6.1 illustrates this algorithm for five MPEG-2 transport stream packets in the interval of 16 isochronous packets. In this case the isochronous packets will be filled with halves of MPEG-2 transport stream packets, so the size of the entities is four data blocks. This means that there are ten entities and the ratio can be calculated as $\frac{10}{16} = \frac{5}{8}$. The table shows for each isochronous packet the value of the accumulator before performing the calculations, the number of entities put into the isochronous packet, and the corresponding number of data blocks.

It is import that the algorithm performs exact calculations, not some rounded floating point arithmetic. This can easily be achieved by multiplying all numbers with the denominator and using integer arithmetic. Instead of using the floor operation, the algorithm has to decide in this case how often the denominator is contained in the accumulator and send this number of packets. If the ratio is less than one, i. e. there are fewer entities to send than isochronous

isochronous packet	accumulator	entities	data blocks
1	0	0	0
2	$\frac{5}{8}$	1	4
3	$\frac{10}{8} - 1 = \frac{2}{8}$	0	0
4	$\frac{7}{8}$	1	4
5	$\frac{12}{8} - 1 = \frac{4}{8}$	1	4
6	$\frac{9}{8} - 1 = \frac{1}{8}$	0	0
7	$\frac{6}{8}$	1	4
8	$\frac{11}{8} - 1 = \frac{3}{8}$	1	4
9	$\frac{8}{8} - 1 = 0$	0	0
10	$\frac{5}{8}$	1	4
11	$\frac{10}{8} - 1 = \frac{2}{8}$	0	0
12	$\frac{7}{8}$	1	4
13	$\frac{12}{8} - 1 = \frac{4}{8}$	1	4
14	$\frac{9}{8} - 1 = \frac{1}{8}$	0	0
15	$\frac{6}{8}$	1	4
16	$\frac{11}{8} - 1 = \frac{3}{8}$	1	4

Table 6.1: Algorithm 6.1 sending five MPEG-2 transport stream packets in 16 isochronous cycles

packets, this operation can be replaced with a simple comparison. The ratio will be bigger than or equal one, if the entities are complete transport stream packets; in this case, each isochronous packet contains an integer number of MPEG-2 transport stream packets. In all other cases, the ratio is less than one. But even in this case the ratio can be reduced to value below one by subtracting the integer number that designates the minimum number of MPEG-2 transport stream packets that have to be put into each isochronous packet. The algorithm then is only used to distribute the remaining MPEG-2 transport stream packets over the isochronous packets. This modification leads to Algorithm 6.2. One property of this algorithm is that the first packet is always a small (or empty) packet, unless all isochronous packets have the same size, and the last packet is always full. If the full packets were rather moved to the beginning of the interval, this might help to reduce late packets. This can easily be achieved by either starting filling packets at the last packet and proceeding to the first one or by initialising *accumulator* with *denominator* $-$ 1 instead of zero. In the demonstration setup, a slightly modified version of this algorithm was used [126].

An alternative strategy would be to fill the first packets of the interval. This would mean to cluster the full packets at the beginning of the interval and the empty (or small) packets at the end of the interval. The advantage of Algorithm 6.2 is a smoother data flow and less buffer requirements at the target, the advantage of the latter approach is the better prevention of late packets. Since the timestamp in the source packet header has to point into the future, this timestamp has to be generated by adding a constant offset O to the theoretical transmission time. This means that a packet received at the gateway at time T has a theoretical transmission time $T + T_0$ (the actual transmission time will jitter a bit around this due to the transmission process) and a timestamp in the source packet header $T + T_0 + O$. In this scenario, it is obvious that an offset $O = 16$ or greater should be sufficient to avoid late packets. The worst case is a packet with a theoretical transmission time T at the beginning of the first cycle of the interval

Algorithm 6.2 Distribution of data blocks over isochronous packets (using integer arithmetic)

```
accumulator ⇐ 0
determine size and number of entities to distribute over isochronous cycles
wholePackets ⇐ ⌊total number of entities / number of isochronous packets⌋ {zero for entities smaller than 8 data blocks}
numerator ⇐ total number of entities mod number of isochronous packets
denominator ⇐ number of isochronous packets
while not all isochronous packets filled do
  accumulator ⇐ accumulator + numerator
  if accumulator ≥ denominator then
    send packet containing wholePackets + 1 entities
    accumulator ⇐ accumulator − denominator
  else
    send packet containing wholePackets entities {empty packet in case of entities smaller than 8 data blocks}
  end if
end while
```

that gets its last data block transmitted during the last (in this scenario the 16^{th}) cycle of the interval. It would, therefore, arrive at the target shortly before start of the next interval $(T + 16)$[6]. If the length of the interval was not 16, all numbers would, obviously, have to be adapted.

This strategy shows two problems. First, since all packets are assigned to fixed transmission intervals and transmission can only be shifted within one interval, but not over the borders of the intervals, suboptimal situations might occur where the algorithm occasionally switches to higher isochronous packet sizes. Especially if the data rate is exactly at or near the value that marks the transition between two packet sizes, it will happen that in some intervals there is one excessive packet that causes this switch to the higher packet size, while the intervals before or afterwards would offer enough empty isochronous packets to take over this excessive data. There is one simple mechanism to avoid this effect, at least if the data is not too bursty. If there is free space available during the current interval to take additional data without increasing the maximum packet size, i. e. if not all packets would be filled up to the maximum size, also the next interval is considered. If the next interval contains more data than the current interval, the algorithm should take data from the next interval for transmission. It has to stop this shifting before it either would have to increase its own maximum packet size or the amount of data remaining for the next interval would be less than the original amount of data in the current interval. The original amount of data in this case means the amount of data available when the algorithm started processing this data, after data was taken to the previous interval, but before data is taken from the next interval. This procedure ensures that a switch to a higher packet size in the next interval is avoided, if possible. As the amount of data shifted is also limited by not draining the next interval under the amount of data in the current interval, it is avoided that too much data is moved between these intervals causing unsteady transmission

[6] In a worst case scenario, the end of transmission could even be about 25 µs after start of the next cycle because asynchronous transmissions can significantly delay the start of an isochronous cycle [8]. Furthermore, some processing time at the target should be taken into account. Therefore, to reliably avoid late packets, the offset O should be chosen at least 0.2 cycles bigger than the interval length.

with unnecessary switches to lower maximum packet sizes. Free transmission time slots for stuffing are transferred from one interval to the next, until they are really needed. Furthermore, since transmission is always shifted to an earlier cycle, this algorithm cannot cause late packets. The only problem this algorithm can cause is increasing the buffer usage in the sink because it has to store some data earlier and keep it for a longer time. However, it turns out that only the buffer usage during transmission of the current interval is increased, but not during the following intervals because during the following intervals this data would be transmitted anyway. And during the current interval, the maximum buffer usage stays below the worst case buffer usage for an interval containing the same number of packets as the next interval. So this algorithm essentially increases the duration of higher buffer usage in the sink, but not the maximum buffer usage that has to be expected for the given maximum number of transport stream packets per transmission period. The amount of data moved between the intervals does, of course, not have to consist of complete transport stream packets, it is also possible to shift single data blocks, but only if if this would not cause a violation of the packetisation rules in one of the affected transmission intervals. So, for example, if one of these intervals requires halves of MPEG-2 transport stream packets, it is, obviously, not allowed to shift a single data block.

The second problem of this algorithm is that the offset O might be too high in some application scenarios. While many systems on the receiving side might offer big buffers that allow storing huge amounts of data, the standard IEC 61883-4 specifies a buffer size of 3264 bytes, which is equivalent to 17 source packets (188 bytes MPEG data plus 4 bytes source packet header) [63]. For full MPEG-2 transport streams with the data rates given in (6.3) and (6.4) that require to transmit isochronous packets of two or three and packets of three or four transport stream packets, this is equivalent to 6.2 and 5.4 isochronous cycles. This means that, for full transport streams, after receiving the isochronous packet, the timestamp in the source packet header should not point more than 6 or 5 cycles into the future. So the algorithm using the numbers given above would perhaps work for partial transport streams with up to one complete MPEG-2 transport stream packet (or slightly more) per isochronous packet. But even this will only work for partial transport streams with almost equidistant timestamps, otherwise buffer overruns at the target would have to be expected[7]. However, for full transport streams, it would no more comply with the standard. For low data rate streams, transmission intervals of at least 8 isochronous cycles are necessary in order to make transmission of eighths possible, longer intervals are even better to ensure a smoother stream. On the other hand, high data rate streams might be handled properly with even smaller interval sizes. Hence, the length of the interval should be adapted with changing data rates. If O is the same as the interval length, it turns out that the maximum buffer usage in the sink will lie in the range between the number of packets transmitted within one interval (for optimal streams with perfectly equidistant timestamps) and twice this value for worst case streams. In this worst case, all the packets of the current interval have a theoretical transmission time near the end of the interval and, therefore, timestamps that reach almost to the end of the next interval. While the buffer is being filled with packets during the next interval, no packets are consumed and, therefore, the buffer has to hold the packets of both intervals. In case of O longer than the interval length, it would even be possible that in the worst (albeit very unlikely) case the buffer could be filled with the packets of three intervals.

[7] For worst case partial transport streams, where the source packets are clustered very much, the timing properties are piecewise similar to full transport streams.

It is proposed that, if the algorithm detects more than eight transport stream packets within one interval, it shall reduce the interval length (to half of the current interval length). According to the considerations above, this ensures that the buffer usage at the sink can be kept below 17, the limit specified in IEC 61883-4. If there are fewer than four packets within the interval, it can be increased again (to double of the current one). The lower limit of packets within one interval (four) could also be chosen smaller in order to introduce a hysteresis and prevent the algorithm from permanently switching between two interval lengths for data rates that lead to interval lengths with packet counts near eight or four. Together with the length of the interval, the system also has to adapt the offset O, so that it reflects the new interval length. In fact, reducing O in case of high data rate streams is the main objective, modifying the interval length is just a requirement for reducing O in order to avoid late packets. Since the time that is added to the reception times in the gateway to generate the source packet header has to stay constant in order to avoid discontinuities of the stream, the sum $T_0 + O$ has to stay constant. Therefore, a reduction of O requires an increase of T_0. While this can be performed rather easily because increasing the overall delay means that there would be some additional empty isochronous packets, the other way is a bit more painful. If the interval length and the offset O are increased leading to a reduction of the delay T_0, this will require some additional packets to be stuffed into the stream at the time of the switch because the theoretical transmission time for given packets is shifted to earlier times, where they would overlap with packets that were scheduled using the lower O. The consequence is that such switch cannot be performed arbitrarily, but it requires some time to stuff the excessive packets into the stream. Since such switch usually occurs together with a switch from a higher data rate to a lower one and, therefore, from a higher maximum isochronous packet size to a lower one, this stuffing can be performed easily by just maintaining the higher maximum packet size for some more cycles in order to transmit additional packets. This means that data is taken from the following cycle as long as the final number of MPEG-2 transport stream packets stays below eight and the size of the isochronous packets stays below the maximum packet size of the former higher data rate. Once one interval contains no packet anymore because they all have been taken to the previous interval, the switch can be performed. It is obvious that this algorithm has an upper limit for O (and for the interval length) because the delay T_0 has to stay positive. This means that O must always be less than the sum of the initial offset O and the initial delay T_0. The initial sum $T_0 + O$ should, hence, be at least 8 in order to allow putting eighths into isochronous packets, but it should be chosen significantly higher to allow smoother transmission of low data rate streams. A natural lower limit for O (and the interval size) is 1. It is clear that limiting the number of MPEG-2 transport stream packets per interval to eight can only work as long as not more than eight transport stream packets per isochronous packet are needed. For higher data rate streams, the buffer according to IEC 61883-4 (which have been designed for streams of up to 60 Mbit/s) is simply not enough. For high data rate streams, more packets per interval are necessary; in general, higher numbers of packets per interval improve the transmission process (make it smoother), but require bigger buffers at the sink devices. It should be mentioned that these changes in the system delay have, of course, impact on the buffer usage in the gateway and, therefore, can influence the mechanism for reconstruction of the reception timestamps.

The algorithm for generating the isochronous packets as outlined above does not care about resource reservation. In fact, in the demonstration setup, no resource reservation is performed, and therefore, the rules of IEEE 1394 are clearly violated. According to IEC 61883-1, the device has to advertise the amount of bandwidth that is required for a stream associated with a certain output plug [58]. If an application tries to establish a connection and initiates streaming, it

will have reserve the required isochronous resources in advance and can only establish the connection, if these resources are available. Although managing these plug control registers might be separated in the software from actually generating the isochronous stream, there must be an interface to the streaming part that is accessed by the part handling plug control register. The streaming part needs the information how much bandwidth has been reserved. Moreover, it has to know the selected data rate (IEEE 1394 support transmission at several different speeds) because it has to send data at this data rate. And the streaming part has to provide information about the required amount of data per isochronous packet. It has to ensure that no isochronous packet contains more data than reserved. This means that if the algorithm above would lead to a maximum packet size exceeding this limit, the algorithm has to delay packets to later intervals or even drop packets if this would lead to late packets. If this happens, the algorithm might go into some error condition and even stop streaming. The better solution would be that it informs the part handling the plug control registers about the increased bandwidth requirements, which in turn could try to allocate the additional bandwidth and adapt the `payload` field of the plug control registers. Once the additional bandwidth has been reserved, the streaming part can start generating bigger packets. The information provided by the streaming part about the required bandwidth should contain some safety margin to make sure that there is enough bandwidth reserved for the case of variations in the required bandwidth. The required bandwidth in the plug control register should not be changed too often. Therefore, it should only be adapted, if the required bandwidth inclusive safety margin differed more than a certain amount from the current information in the plug control register. But this question of policy is not discussed further here.

6.2 Audio Gateway between IP and IEEE 1394

A very interesting area for multimedia streaming are gateways that connect to the Internet. The Internet is an extensive source for all kids of information, among them also multimedia streams. As already described in previous chapters, the IP protocol is not very well suited for real-time multimedia transmission. Nevertheless, there are real-time multimedia streams available. For example, many radio stations offer live streams of their programs. These streams are typically played on personal computers, but consumer electronic devices are usually not able to play them. Of course, there are more and more consumer electronic devices equipped with connections to the Internet, and eventually they will be able to directly play such streams. There are consumer electronic devices with digital network connections, for example the Sony LISSA HiFi set, which consists of devices connected with IEEE 1394. These devices can be used as sources or sinks of audio streams according to IEC 61883-6 [62]. Unfortunately, they cannot use IP based protocols. It is not possible to play an audio stream offered in the Internet on an IEEE 1394 enabled audio amplifier—unless it is also part of the IP network and understands IP based streaming protocols. Therefore, it makes sense to try to interconnect those networks. Figure 6.6 gives an impression of the situation: On the one hand, there is the Internet with (among many other services) a few radio stations offering live streams, and on the other hand, there is a typical IEEE 1394 bus providing—among other consumer electronic devices—an audio amplifier. And between those networks, there is a gateway [61]. The aim of the present work is to make multimedia content that is available in one network within the home also available in the other one. Here it is to make content from an IP network available on IEEE 1394. For this purpose, a kind of interconnection gateway is needed. It is possible to use one gateway that

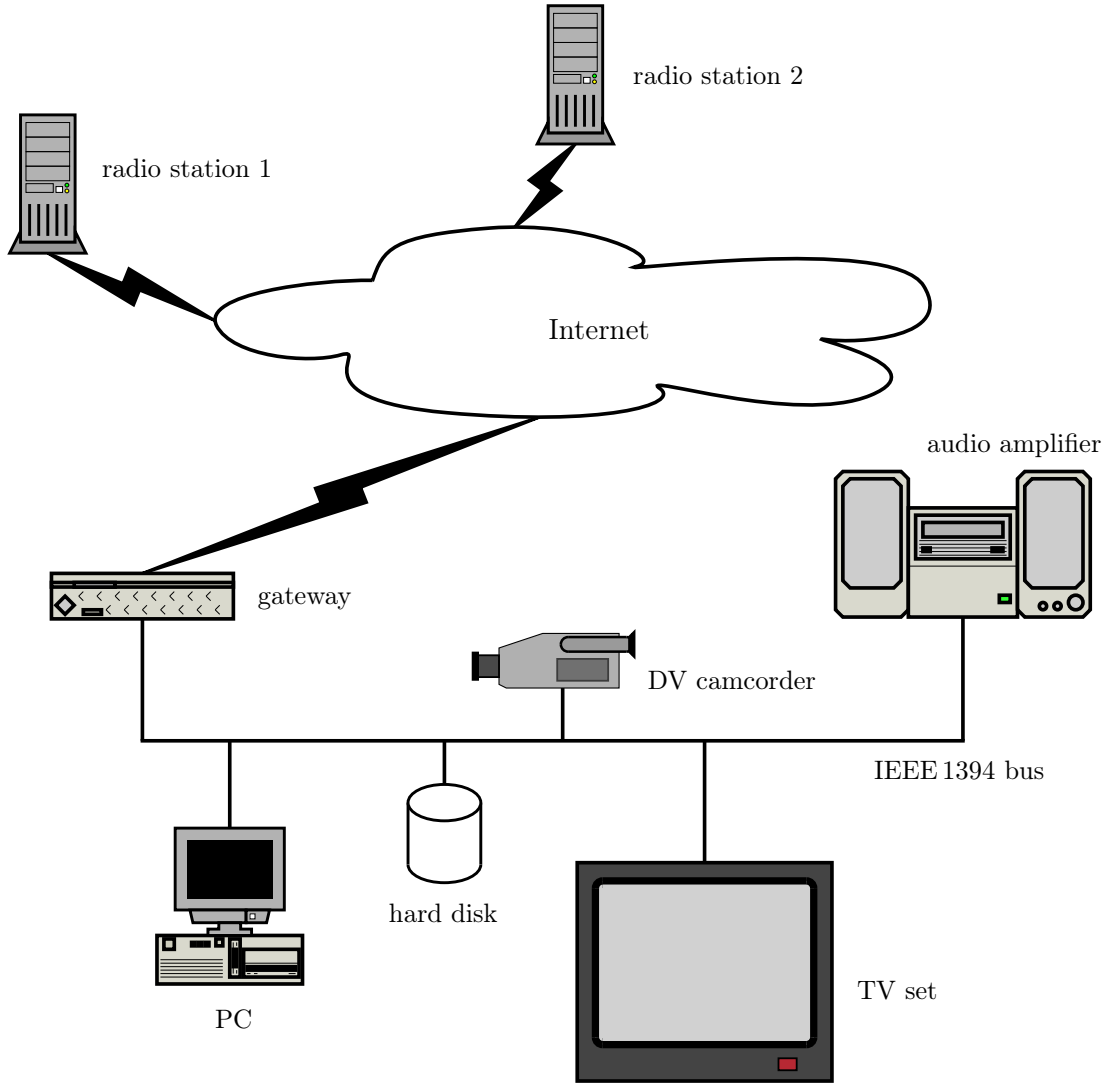

Figure 6.6: Setup of the gateway between Internet and IEEE 1394

offers many services of the Internet. But of course, it is also possible to have more than one gateway, each offering some services that are available in the other network. A service might also be offered by several gateways.

The design of the gateway has to take into account that on both networks data shall be transmitted in a form which is suitable and common on that particular network. It should not just retransmit data in a format that is used on the source network, but unusual on the target network because then usual consumer electronic devices would not be able to deal with that data. Hence, according to the classification in Section 5.1, this gateway is a data format layer gateway.

To achieve real convergence between IP networks and IEEE 1394, it is necessary to either have one universal gateway with many input and output modules or to have many different kinds of gateway between IP networks and IEEE 1394, one for plain audio, one for video, one for combined video and audio, and so on and also for both possible directions. The example

here focuses on the very special case, which is illustrated in Figure 6.6.

Many radio stations provide live streams of their programs on the Internet. The gateway described here should make these streams available on the IEEE 1394 bus. This would enable an amplifier connected to the IEEE 1394 bus to play that stream. Unfortunately, most of those stations use some proprietary format (e. g. RealAudio). For the implementation it was assumed that radio stations provide their streams within RTP and use some data format that can be decoded. The implementation described below uses an external program to receive and decode audio streams from the IP network, and therefore, all formats understood by this program are supported by the gateway. In fact, since this external program also handles the transport protocol, this is not limited to RTP, but other protocols, like for example RDT, are supported as well.

The gateway must perform the following tasks:

- It must handle control information from both sides.
- The (audio) stream must be converted from RTP on the IP based network to IEC 61883-4 on IEEE 1394. This involves two parts:
  - conversion of the data format
  - reconstruction of timing information

The actual implementation, which is described in [135], focuses on the streaming part. The control part was not implemented, and hence, the following description of the control part describes how an added control part would work. However, it has to be stressed that there have been implementations of streaming applications by the author of this dissertation where, in order to start streaming on an IEEE 1394 bus, an external program is called after setting up the connection appropriately according to the HAVi protocol on IEEE 1394. It has not been coupled with this particular streaming gateway, but it has been proven that such mechanisms work and are not just theoretical considerations. In these applications, the streaming software is started in a child process that is controlled via two pipes for standard input and standard output. Whenever connections are established or broken on HAVi, an appropriate command is sent to the streaming application on the standard input pipe, and a status response is returned on the standard output pipe.

The gateway was implemented as a multithreaded user space application running under the operating system Linux. It uses the IEEE 1394 subsystem of the Linux kernel via the library `libraw1394`. The code for packetising and transmitting isochronous packets is based on the library `libiec61883`. This library, `libiec61883`, could not be used directly for transmission on IEEE 1394 because it does not allow to influence the packetisation (the number of samples per isochronous packet) and, finally, transmission rate of samples. It would be strictly coupled to the IEEE 1394 isochronous cycles and the nominal sample rate. This is a possible solution for playing back local files, but the gateway has to adapt the output sample rate. It has to match the incoming sample rate rather than the IEEE 1394 cycle time. Therefore, the code of `libiec61883` was integrated into the gateway application and adapted appropriately.

The application consists of three threads, which are running in parallel:

- One thread running the decoder is in charge of receiving and decoding the audio stream. It puts the PCM samples into an audio buffer.

- Another thread is responsible for transmission of data on IEEE 1394. It reads samples from the audio buffer, creates isochronous IEEE 1394 packets, and transmits them.
- A third thread is used to output status information about the operation of the gateway. It maintains a graphical view of the state of the system. This thread is not essential for operation, it could be removed.

6.2.1 Control

On the IEEE 1394 bus, some middleware protocol has to be used to control the devices on the bus. Within this concept, HAVi [60] is used, but of course, the implementation for other protocols like AV/C [59] or UPnP [77] could be performed in an analogous way. Therefore, the gateway will appear as a HAVi device on the IEEE 1394 bus and will supply a tuner FCM (functional component module). This tuner FCM provides some radio programs (the URLs of the offered services might be taken from some configuration file or found by some discovery mechanism that is outside the scope of this document) and makes them public via service lists. Figure 6.7 illustrates this mapping. When a service is selected (by issuing the appropriate HAVi command), the gateway will establish the RTP connection by use of RTSP [74] and start buffering (see Subsection 6.2.3). When the corresponding IEC 61883 connection is established by appropriate manipulation of plug control registers, it will start streaming on IEEE 1394. The service locators used by the tuner FCM could be the URLs.

The streaming part that has been actually implemented is a console based stand-alone program [135]. Once it is started, it commences buffering audio data and later on transmission on IEEE 1394. This application has to be extended in a way that it can be controlled with simple commands from standard input. The minimum set of commands to be supported should include:

- `exit`: Exit the program.
- `receive <URL>`: Start receiving, decoding and buffering the stream identified by the given URL.
- `receive stop`: Stop receiving.
- `transmit <channel>`: Start transmission of audio stream on the given isochronous channel. If there is no audio data available, transmit silence.
- `transmit stop`: Stop isochronous transmission.

Furthermore, the application might support in addition commands like `status` to display status information, `debug` to switch into a debugging mode, `version` to display version information, and many more. Of course, the names of the commands are not important, the mentioned names are just examples, it would also be possible to provide the same functionality with different command names. After each command, the application shall print a response to standard out indicating whether the command has been executed successfully or not.

Such application can be integrated into HAVi software easily. In the EU funded project HOME-PLANET [2], where the task of the Institute of Computer Technology was the implementation of the HAVi software and the graphical user interface, exactly such integration was realised. In this case, a command line based open source application called `mpeg2ts1394dec` implemented by the author of this dissertation is used by the HAVi system. This application

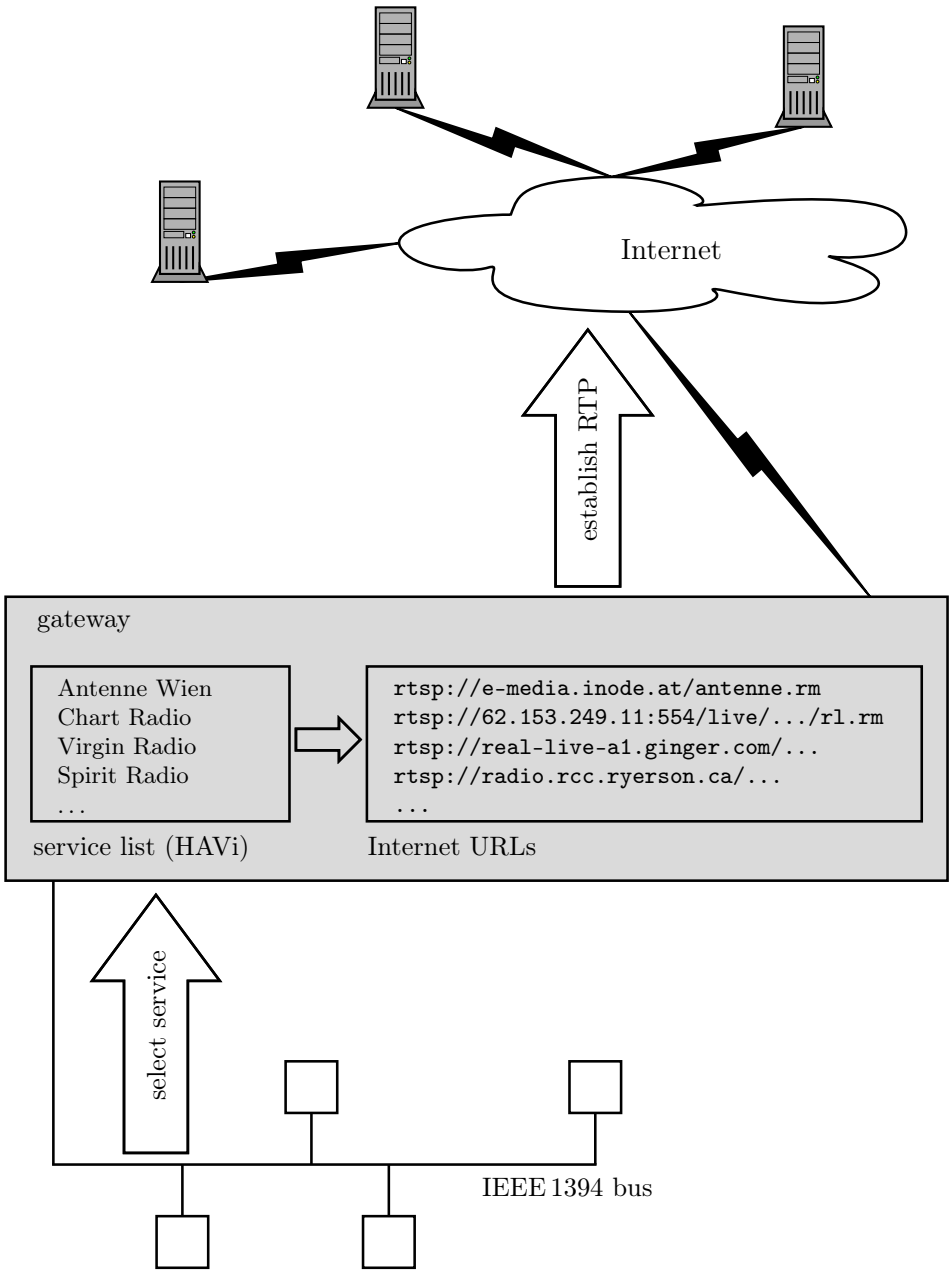

Figure 6.7: Mapping between services (in the HAVi network) and URLs (in the Internet) within the gateway

can receive an MPEG-2 transport stream over IEEE 1394 according to IEC 61883-4, extract one TV or radio program, decode it, and present it via graphic card and sound card. The HAVi software forks a separate process, and within the context of this child process `mpeg2ts1394dec` is executed. Communication with this process is performed via two pipes, one for standard input and one for standard output. The software interface for controlling this decoder application consists of three functions, one to start the child process, one to terminate the child process, and one to send commands to the child process. One advantage of this approach is

the very clear separation between the applications, it is not necessary to integrate or link any code from the child program into the HAVi application. This can also prevent problems with software licenses in some cases because, for instance, some open source licenses do not allow to link their code with code under another, in particular commercial, license.

In case of the audio gateway discussed here, the `receive <URL>` command has to be issued to the child application, when the tuner FCM gets a `SelectService` call. Once all the necessary connections have been established, the isochronous connection according to IEC 61883-1 in the output plug control register as well the internal connection between the DCM plug and the FCM plug of the tuner FCM, the `transmit <channel>` command has to be issued. If a connection gets broken, the `transmit stop` command has to be issued.

There are two possibilities for a system that supports reception and transmission of several streams at the same time. There might be either several independent FCMs, each able to receive on stream, or on tuner FCM with several output plugs. Although both approaches are possible, the more proper solution would be to use two separate FCMs because a tuner normally is able to receive only one stream. Several plug are typically used by tuners that receive multiplex streams and are able to provide different sets of components at different output plugs. Hence, if several streams are to be supported, several instances of the tuner FCM should be instantiated, each forking its own child process running the streaming application. However, it should be noted that the IEEE 1394 subsystem limits the number of isochronous streams that could be transmitted at the same time. Therefore, it does not make sense to have more instances of this streaming application running in parallel because they would not work properly. Consequently, also the number of tuner FCMs would be limited. One possibility would be to have a lot of independent tuner FCMs, which can receive many streams at the same time, but only allow a smaller number to transmit the received program on IEEE 1394. This could be realised by limiting the number of DCM output plugs. The upper limit for the number of simultaneously received streams depends on the network connection to the Internet. But it is questionable, whether it makes much sense to have more tuner FCMs than isochronous channels that can be operated at the same time.

One clear limitation of this approach, where the streaming part is sourced out into a separate application that only communicates with the rest of the HAVi software via a simple text oriented control protocol and only offers transmission via IEEE 1394, is that the destination of the stream always has to be the IEEE 1394 bus. It is not possible to establish internal connections with, for example, an amplifier FCM or a disc FCM for local playback or recording. One consequence is that it might make sense to establish a fix connection between FCM output plug and DCM output plug. In order to allow such features, like internal streaming connections between different FCMs, the streaming part of the software would need to be coupled more tightly with the rest of the HAVi software. However, for just providing the gateway functionality, which is the topic of the present thesis, this solution would work perfectly.

6.2.2 Conversion of Data Format

An audio stream on IEEE 1394 compliant to IEC 61883-6 [62] usually contains uncompressed audio. This is viable because on that bus within the home there is usually enough bandwidth available and it makes it easier for the amplifier to play the stream since no decoding is necessary. On the other hand, the RTP stream arriving from the Internet might as well contain uncompressed audio as defined by [79, 80], but due to limited bandwidth in public networks, it will usually utilise some compressed format, e. g. MPEG as outlined by [81] or some proprietary

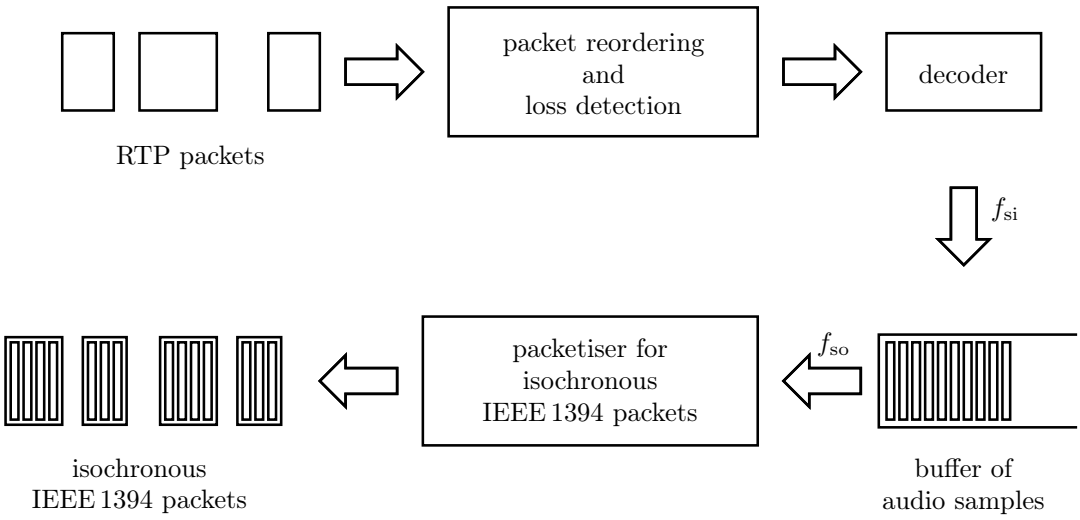

Figure 6.8: Processing and buffering of audio data in the gateway

format. Therefore, some decoder (as shown in Figure 6.8) is needed to obtain the plain audio samples.

The most common audio data formats used in the Internet, e. g. RealAudio or WMA (Windows Media Audio), are proprietary data formats, the specifications are not freely available. Consequently, it is not possible to build a decoder, but some commercial decoder software has to be used. It was decided to use the `mplayer` application for decoding these data formats. `Mplayer` is an open source application that integrates a wide variety of binary decoder modules from many vendors as well as open source decoder modules. It, therefore, supports reception of many different data formats over different transport protocols. Unfortunately not all combinations of transport protocols and data formats are supported. In particular, it turned out that, at least with the version used within the project, it is not possible to receive RealAudio in UDP packets. It is only possible to receive them in RDT packets within TCP. It has already been emphasised that TCP is no proper protocol for multimedia streaming for several reasons. The `mplayer` code is used to receive the data, interpret the transport protocol, decode the data format, and deliver plain audio samples. To perform this, one of its output module that is usually used to store the PCM samples into a file has been modified to hand over the samples to the IEEE 1394 transmission module.

6.2.3 Reconstruction of Timing Information

The RTP data stream contains a timestamp in the RTP header of each packet. This timestamp is intended for two types of synchronisation purposes. In conjunction with the mapping between NTP timestamps[8] and RTP timestamps contained in the sender's report (SR) of RTCP [73], it can be used for inter-media synchronisation, which usually means ensuring an audio stream to be lip sync with a video stream. Since here only single streams[9] are used, this kind of

[8] The NTP timestamps used in RTP refer to wallclock time (absolute time).

[9] Even if a multiplexed stream (e. g. an MPEG-2 transport stream containing audio and video) was transmitted via RTP, that synchronisation mechanism would not be not used because it is only one single RTP data stream. The synchronisation between video and audio is done by the decoder (exactly: the demultiplexer) in this case.

synchronisation is not needed.

Intra-media synchronisation is the other issue, and this has to be performed always. As already lined out in Section 4.1, it essentially consists of three parts:

- A drift between the sampling clocks at source and sink must be compensated.
- The arrival jitter of RTP packets must be handled.
- Lost packets and silence periods must be treated correctly.

In an RTP data stream, the RTP timestamps could, in principle, be used to eliminate a drift between the sampling clocks. That timestamp reflects the sampling instant of the first octet in the RTP data packet [73]. However, using these timestamps together with a local reference clock requires that these reference clocks at source and sink must be synchronised. Otherwise, the drift of the clocks between source and sink leads either to buffer underruns or overruns at the sink device. Of course, this synchronisation can be achieved, e. g. by using protocols like NTP (network time protocol) [136]. But it is also possible to use a much simpler approach and send the samples using the average arrival rate f_{si}. This method is explained in detail below. This can be done because no inter-media synchronisation is necessary in this case where only a single audio stream is to be transmitted.

The same applies for the arrival jitter. Instead of working with timestamps, it is easier to maintain a buffer, that outputs the samples at the average arrival rate f_{si}. This, of course, requires that the samples have to be played at a constant rate, i. e. the time between two samples must be constant. For audio, this is fulfilled, but there are other kinds of stream that do not meet this requirement.

Discontinuities in the RTP timestamps might be caused by lost packets (in this case there would also be a discontinuity in the sequence numbers) or silence periods (which are usually marked by a special bit in the RTP header). In case of radio, silence periods are not usual, they are rather used in applications like audio/video conferences. In both cases, an appropriate number of empty (or interpolated) samples must be inserted into the buffer, so that the target device outputs silence. This number can be determined by calculating the difference between the timestamp in a new packet and the expected timestamp, which is the timestamp of the last packet increased by a number corresponding to the duration of the audio signal in the last packet. If no packets arrive for some time, then empty samples must be created as well to avoid a buffer underrun.

The gateway starts filling the buffer (see Figure 6.8) with audio samples after the connection has been established and RTP packets arrive. It must queue samples for at least the expected maximum arrival jitter, but the delay should be small enough to make it acceptable to the user. Therefore, a buffer size in the range of one or two seconds might be a good compromise. Then the gateway calculates the time $T_{\mathrm{so}} = 1/f_{\mathrm{so}}$, which should elapse between two samples (in units of the IEEE 1394 cycle time, i. e. 1/24.576 MHz which is approximately 41 ns), either by using the nominal sample rate or the measured arrival rate f_{si}. Then, after starting with some cycle time for the first sample, it is possible to calculate the cycle time for each sample, when it should be presented at the receiver. It is very important to note that the cycle time is synchronised between all nodes on the bus by IEEE 1394. Therefore, this time is the same at the source (i. e. at the gateway) and the sink device, there is only a small, but constant, offset in-between, there is no drift. That calculated presentation time gives a hint how to packetise the stream on IEEE 1394: Roughly, all samples whose cycle times yield the same cycle count are

put into one isochronous packet. More precisely, all samples with presentation times within one 8000 µs interval, which corresponds to 3072 units of the IEEE 1394 cycle time, have to be put into one isochronous packet. It is not really necessary that the cycle counts of the presentation times are the same, the only important thing is that the interval correspond to one isochronous cycle. Since the audio signal has equidistant samples, this algorithm ensures that all the isochronous packets have approximately the same number of samples, only varying one single sample. Therefore, it is not necessary to employ a more sophisticated algorithm for distributing the samples over the packets. For some samples, IEC 61883-6 requires timestamps in the `SYT` field of the CIP header. For this purpose, the lower 16 bits of the calculated cycle time are used. Those packet can be determined by incrementing a counter for each sample, and when it would reach `SYT_INTERVAL`, the counter is reset to zero and the `SYT` field has to be filled with the presentation time for this sample. This algorithm leads to a transmission pattern that is called "non-blocking transmission method" in IEC 61883-6 [62]. The number of samples within one isochronous packet varies, and the transmission timing of the samples matches the presentation timing very well. The other transmission method, the "blocking transmission method", where either a packet containing a fixed number of samples (typically `SYT_INTERVAL` samples) or no samples at all, could be implemented easily as well. But this transmission method would only make sense for limited devices that are only capable of transmitting isochronous packets of a fixed size. On powerful devices like personal computers, which are used as platform for this gateway, there is no reason for using the blocking transmission method.

One important design choice is at which cycle to start transmission or, the other way round, for a fixed start cycle, which presentation timestamp to use for the first sample. IEC 61883-6 specifies a default transfer delay as 354.17 µs + 125 µs [62]. Consequently, the timestamp should point almost 4 cycles into the future.

As already mentioned, in the actual implementation, the existing open source application `mplayer` is used to receive the stream and extract the PCM samples [135]. One drawback of this solution is that the gateway software does not get the samples as soon as they have been decoded, but `mplayer` buffers bigger chunks of data and delivers them at once. Of course, the best would be to decode each packet received from the IP network and forward it immediately to the gateway software, so that it gets accurate information about the timing of the incoming stream, ideally together with timestamp information. The size of these chunks of data, which `mplayer` forwards at once, can be configured. The default chosen for this implementation is 16384 bytes. For typical audio streams (as used on audio CDs: 44100 Hz sample rate, stereo, 16 bit/sample), this corresponds to approximately 100 ms. During the tests, this proved to be a good default value.

The gateway starts transmission based on the nominal sample rate with the IEEE 1394 cycle time as reference. As already emphasised, this transmission rate will not fit the transmission rate at the source, unless the reference clocks are synchronised. Consequently, data will either accumulate in the buffer of the gateway, or the buffer will run out of data. If the buffer of audio samples in the gateway tends to fill more and more in the long-time average, the gateway transmits too slowly on the IEEE 1394 bus. Therefore, it should slightly decrease T_{so}. Vice versa, it should slightly increase T_{so} if the buffer seems to run out of data. For adaptation of T_{so} to make it as smooth as possible, a more or less sophisticated algorithm can be used. For the implementation of the gateway, it was decided to introduce a modifying factor for the output sample rate f_{so} that is proportional to the deviation of the buffer usage from the target buffer usage, which is 50 %. Due to the bursty arrival process over the IP network and the bursty forwarding of decoded data from the `mplayer` application, the buffer usage oscillates quite a

lot. In order to prevent these variations from directly affecting the output sample rate f_{so} and, consequently, causing variations in the output signal frequencies, the algorithm does not directly take the buffer usage, but an exponentially weighted average over the last measured buffer usages. The average buffer usage (in percent) is calculated according to (6.12):

$$AverageFillstate(t) = AverageFillstate(t-1) \cdot w + CurrentFillstate(t) \cdot (1-w) \quad (6.12)$$

The weight w determines how much the old buffer usages influence the average. If it is chosen 0, the average buffer usage $AverageFillState(t)$ will always be the same as the current buffer usage $CurrentFillstate(t)$, so no averaging will be done at all. On the other hand, if it is chosen 1, than $AverageFillstate(t)$ will always stay at its initial value $AverageFillstate(0)$, which is obviously not useful. Since transmission starts with a half filled buffer, the initial value $AverageFillstate(0)$ is chosen 50 %. As default weight $w = 95\,\%$ was chosen. This means that the current value has a weight of 95 %, and older values have exponentially decreasing weights. Only approximately the 30 most recent values have significant influence to the calculated average and together form more than 75 % or the result. This value leads to good results in the test environment. However, since the reception parameters, in particular the interarrival jitter of the packets and the packet sizes, can vary significantly, all these parameters, which are used to smooth the reception process, might have to be adapted. It was not evaluated which values of w would be the best in which scenarios. Bigger values of w, near 1, have the advantage that the calculated average buffer usage is very smooth and fluctuations in the reception process are attenuated very good. The drawback is that it can only adapt very slowly in case of changes and the calculated average could differ very much from the actual current buffer usage. On the other hand, very small values of w, near 0.5 or even lower, have the drawback that fluctuations in the reception process are hardly attenuated and, more or less, directly affect $AverageFillstate(t)$. The advantage is that the algorithm can adapt very quickly in case of changes in the reception process.

The factor that modifies the output sample rate f_{so} is calculated as:

$$factor(t) = 1 - [1 - 2 \cdot AverageFillstate(t)] \cdot MaxDeviation \quad (6.13)$$

In this formula, $MaxDeviation$ designates the maximum relative deviation from the nominal sample rate that could be handled by the algorithm. It is a linear function that yields results between $1-MaxDeviation$ and $1+MaxDeviation$ for buffer usage ($AverageFillstate$) between 0 and 1. For example, if the system should be able to cope with sample frequencies varying up to 1 % from the nominal sample rate (with respect to the IEEE 1394 cycle time as reference clock), it has to set $MaxDeviation = 0.01$. In this case, an empty buffer would yield $factor(t) = 0.99$, whereas a full buffer would yield $factor(t) = 1.01$. The output sample rate is calculated by multiplying the nominal sample rate with this factor:

$$f_{so} = f_{nominal} \cdot factor(t) \quad (6.14)$$

It is obvious that a too low output sample rate f_{so} (i. e. lower than the input sample rate f_{si}) leads to a buffer filled more than 50 %, which leads to a factor greater than 1, slowing down the growth of the buffer usage and, consequently, of $factor(t)$ itself. At some point, a steady state will be reached, where $factor(t)$ corresponds to the deviation of the input sample rate f_{si} from the nominal sample rate $f_{nominal}$ and the (average) buffer usage stays almost constant. In this state, the output sample rate f_{so} is the same as the input sample rate f_{si}. In the test setup, a

default $MaxDeviation$ of 2.5 % was used. Obviously, if the actual deviation of the incoming sample rate f_{si} from the nominal sample rate f_{nominal} reaches or even exceeds $MaxDeviation$, the system will not be able to handle this situation and either experience buffer underruns or overruns. One drawback of this algorithm is that the steady state buffer usage depends on the input stream, and therefore, also the overall delay of the system is not constant. A solution to this problem could be real measurements of the (average) incoming sample rate f_{si} instead of this indirect measurement via the buffer usage. Since in case of a transmission process that is only based on the measured (average) incoming sample rate f_{si} small calculation errors still lead to buffer overruns or underruns, it is necessary to have an additional correction factor that is based on the buffer usage that drives the buffer usage back to the target (50 %). But in this case, the buffer usage will always stay near the target with just small deviations, which are independent of the actual incoming sample rate f_{si}. A detailed evaluation of such algorithm, where the transmission rate is based on the measured arrival rate together with modifying factor dependent from the buffer usage, is given in Subsection 6.1.2. The scenario here is quite similar. There is a known theoretical arrival rate, here it is the nominal sample rate, in the other scenario it is a theoretical MPEG-2 transport stream packet arrival rate. In both cases, the entities arrive with equidistant time intervals between them, and in both cases, the underlying hardware and software layers provide this data as bigger chunks.

Nevertheless, the implemented audio gateway modifies the output sample rate f_{so} only based on the buffer usage. This measurement is always performed after sending a configurable number of new isochronous packets, by default it is set to 1000. Hence, on average, 8 measurements per second are performed. In order to avoid too many changes of $factor(t)$ and, as consequence, of the output sample rate f_{so}, the gateway has a configuration parameter for a minimal deviation of the buffer usage from the buffer usage at the last synchronisation action that can cause a new synchronisation. In other words, if the buffer usage differs less than this parameter from the buffer usage at the last synchronisation, no new calculation of synchronisation parameters will be performed. This parameter clearly reduces the number of adaptations. However, it has the drawback that this causes fewer adaptations, but with bigger modifications of the output sample rate f_{so}. It can generally be considered better to have more, but less intrusive modifications. Therefore, it is best to set this parameter to zero. For other values, this parameter is proportional to the size of the steps in the modification of the output sample rate f_{so}.

6.2.4 Evaluation

This gateway was tested in different scenarios. These different sources for audio data were used:

- Audio files stored locally on the hard disk of the gateway were one source of audio. When playing such files, the system can, of course not be seen as gateway between different networks, but since `mplayer` supports playback of local files, this was a convenient method for testing.
- There were MP3 (MPEG-1 layer III) files stored on another PC within the local network. On this PC, there was a primitive RTSP server from the LIVE555 project that can stream MP3 files over RTP/RTCP.
- A second test server in the local network used the free RTSP/RDT server `Helix`. This server provided RealAudio files.

- Furthermore, the gateway was tested with live streams from radio stations in the Internet. These radio stations support RTSP and as transport protocol only RDT. Due to incompatibilities with the server software, it was only possible to receive these streams over TCP. This streams provided bit rates in the range of 32 kbit/s to 64 kbit/s.
- In addition, a live stream of a radio station in the Ogg Vorbis format over HTTP was used for testing. The bit rate of this stream was 160 kbit/s.

As sink for the audio stream on the IEEE 1394 bus two different devices were used.

- A Linux based computer running the open source application `amdtpplay` was connected to the IEEE 1394 bus. This application, which is described more in detail in Section 6.3, receives an audio stream according to IEC 61883-6 over an IEEE 1394 interface card and outputs the stream on its local sound card.
- In order to test the gateway with real consumer electronic devices, a Sony HiFi set called LISSA was used. In particular, the component Sony STR-LSA1 that contains a stereo audio amplifier (besides an FM/AM stereo radio receiver) was used. It is connected to the IEEE 1394 bus and has two analogue loudspeakers.

It should be mentioned that testing with the PC based application was rather easy because the `amdtpplay` application can just be told to listen to a particular isochronous channel, and then it will start receiving. The Sony LISSA HiFi set, however, has no user interface allowing to connect to some isochronous channel. For that purpose, the control protocol AV/C including the connection management procedures of IEC 61883-1 has to be used, but this part was not implemented in the gateway software. Experiments with manually crafted AV/C command frames were not successful. It seems that Sony uses some proprietary extensions to the AV/C protocol, but evaluating them was beyond the scope of this project. The workaround was starting playback from the CD player component of the HiFi set LISSA, which is called CDP-LSA1, via the user interface on the LISSA components and, that way, causing the components to exchange the commands necessary to setup a streaming connection, i. e. the internal connections within both AV/C units as well as the isochronous connection via IEEE 1394 between them. The stream from the CD player component was stopped by manipulating its output plug control register (setting the point to point connection counter to zero). Since the amplifier component STR-LSA1 stayed active listening to the isochronous channel and playing back the stream, the gateway could then start transmitting on this channel, and this stream could be listened to via the loud speakers. It is clear, that such "hijacking" of an isochronous connection is no proper solution. It is, of course, necessary to make the streaming part of the gateway interoperate with a control part and establish isochronous connections using the protocols designed for this purpose. However, for testing purposes, this worked fine.

The conclusion of these tests is that the gateway works fine with both, the consumer electronic device LISSA as well as in case of playback on a PC. Since it was possible during some tests to vary the speed of the audio streams and, consequently, modify the output sample rate, it was possible to test the behaviour in case of slower or faster playback. On the PC based playback device, this causes glitches because the sound card is set to the nominal sample rate, and if the stream arrives faster or slower, it has to drop or insert additional samples. But also on the LISSA device, there were glitches in case of bigger deviations from the nominal sample rate, albeit they were less annoying than in case of the PC based playback. This only shows, that also the LISSA device is only able to cope with sample rates inside a rather narrow band around the nominal sample rate.

6.3 Emulation of Audio Gateway between IEEE 1394 and MOST

Since no dedicated MOST hardware was available for this work, the MOST bus was emulated using a PC sound card, which represents the key characteristics of MOST from the point of view of the software implementing the streaming part of the gateway. The base idea of this emulation is the fact that MOST is a synchronous bus where the timing of the DAC at the sink device is directly coupled to the bus timing, which is enforced by the timing master. Therefore, the main difference to the gateways described in the previous sections is the fact that it is not possible for the gateway to maintain the exact timing of the original stream. It is not possible to couple the clock at the sink DAC with the clock of the source of the stream. In this regard, a speaker system connected to a MOST bus has the same properties as a local sound card from the point of view of the gateway. Sound cards can be set to some discrete sample frequencies, which are usually generated by manipulation (e. g. division) of the frequency of a local oscillator on the sound card. Sound cards are usually more flexible then the MOST system with regard to the supported sample frequencies. They typically support a more or less wide (depending on the hardware and the implementation of the driver) range of sample frequencies, whereas the speakers on the MOST bus can only cope with one single sample frequency. However, this is only a minor difference. And since MOST was designed for multimedia transmission, especially with audio in mind, it usually uses the most common audio sample frequency, which is 44100 Hz. So in this emulation, the MOST bus is imitated by output on the local sound card.

The gateway described here works with audio streams of a sample rate at or near of the sample rate of the sound device. If the (nominal) sample frequencies of the audio stream and the playback device differ too much, some resampling algorithm must be used. Such resampling algorithm was not implemented here, but it would be no problem to implement one. In principle, this could be done by repeating or dropping existing samples. It is obvious that this would not be the best solution. In case of a higher target sample rate that is an exact multiple of the source sample rate, it would be possible to repeat each sample according to the ratio of the sample rates. However, it would be better to generate the additional samples not just by repeating the existing ones, but by interpolating them in order to make use of the higher available sample rate and reduce high frequency noise. If the target sample rate was not an exact multiple, then even some of the original samples should not be passed through directly because their sampling instant would not be aligned exactly with a sampling instant at the target rate. In case of lower target frequencies the source frequency is an exact multiple of, it would be possible to pass through the according fraction of the samples and drop the other ones. However, since this can cause aliasing frequencies, a low pass filter should be applied before dropping the samples. The best and cleanest form of transformation would be reconstruction of the analogue signal, applying low pass filtering at half of the target sample rate according to Shannon's sampling theorem [22], and sampling this signal at the target sample rate. This, of course, also introduces some difficulties, and optimisations are possible, but this is outside the scope of this work. Digital sample rate conversion is a well-known problem in digital signal processing. Several algorithms exist, more information can be found in literature [137]. In the present example, where the Linux sound architecture ALSA is used, this system would even provide a software layer that performs this resampling for sample rates not supported by hardware. So for this problem of sample rate conversion, there are existing solutions.

The demonstration setup is shown in Figure 6.9. The system receives an audio stream from the IEEE 1394 bus according to IEC 61883-6. In the demonstration setup, the source of these streams is one component of the Sony LISSA HiFi system, either the radio tuner or the CD

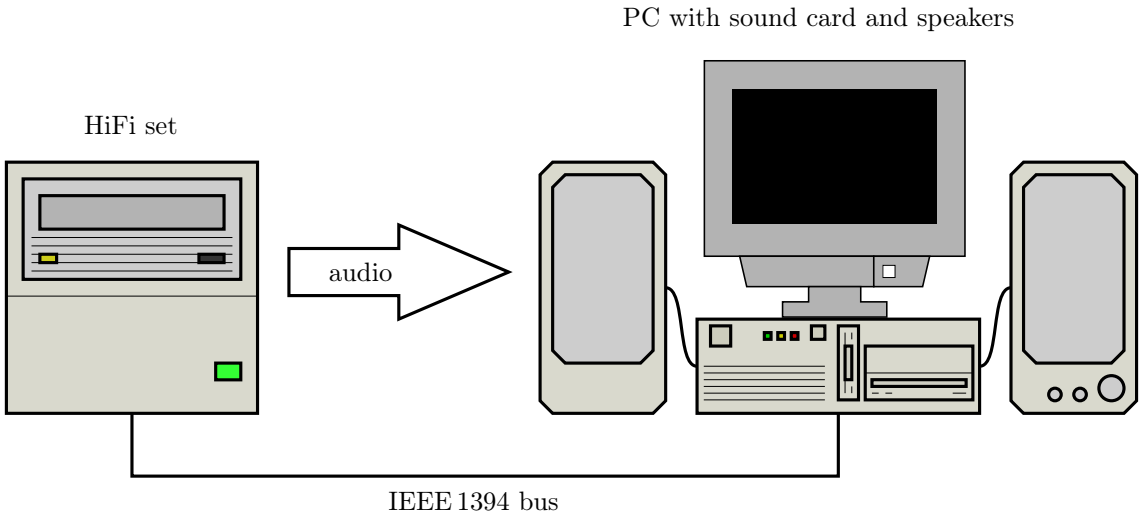

Figure 6.9: Demonstration setup for emulation of a gateway between IEEE 1394 and MOST

player or the minidisc player. These streams consist of plain PCM samples at a sample rate of 44100 Hz and two channels (stereo). The software was mainly developed during practical courses at the Institute of Computer Technology of Vienna University of Technology under supervision of the author of this dissertation, who also revised and significantly improved the result. It was implemented on a PC running Linux as operating system. To access the IEEE 1394 bus, it uses the library `libraw1394`, which has already been described in Subsection 6.1.2. It can work with any OHCI compliant IEEE 1394 card. For access to the audio hardware, the latest version uses the ALSA sound system (standard in Linux 2.6) [138], but it can also work with the Open Sound System that was used in Linux 2.4 and before [139].

The main functionality of the system consists of getting isochronous packets from IEEE 1394, extracting the audio samples, and feeding them into the audio system. Getting the packets from IEEE 1394 is a very easy task. On initialisation some parameters like maximum packet size and buffer size have to be set, and then the library `libraw1394` delivers the isochronous packets via a callback function. In IEC 61883-6, the `FN` (fraction number) field, which determines into how many data blocks each source packet is split, is always 0. This means that source packets are not split and the terms source packet and data block are equivalent. One data block contains all data belonging to one "event", i. e. one audio sample period. Each data block contains one quadlet per channel, so for stereo audio, it contains two quadlets. It carries data compliant with IEC 60958, consisting of (up to) 24 bit plain audio samples plus one byte header information. IEC 61883-6 supports more audio data formats, but the implemented software only supports this data format. Since the audio system supports only 16 bit samples, the least significant bits of each sample are truncated, then the bytes have to be swapped in order to accommodate different endian conventions. Finally, this data is written to the audio system. The system also detects changes in the audio parameters—like sample rate or channel number—and adapts configuration of the audio system accordingly. It also detects interruptions in isochronous transmission and handles them appropriately.

Since audio data consists of plain PCM samples without encoding on both sides (IEEE 1394 and the audio system as imitation of a MOST bus), this is an example of a transport layer gateway. If the IEC 60958 format was seen as a special data format and getting the audio data

(discarding the 8 bit header and truncating it to 16 bit by discarding the least significant bits) as kind of decoding, it could also be seen as a data format layer gateway with a very thin decoding layer at the source side. If audio data was encoded in some more complex format on the source side, this thin decoding layer could easily be substituted with with an appropriate decoder leading to a real data format layer gateway.

This main functionality alone would not be enough. The problem is that the audio system is configured to some sample rate (the nominal sample rate of the received stream) and consumes data according to a local clock integrated into the audio hardware. There is an audio buffer, which is filled by the application and drained by the audio hardware. If the audio hardware consumes data faster than it is delivered, the buffer will eventually underrun. On the other hand, it will overrun if the audio hardware does not consume data fast enough. The size of this buffer differs for different hardware or drivers, but for the sample rates used in this setup, it can typically hold data for several hundred milliseconds.

Both, buffer overruns and underruns lead to audible artefacts. Buffer underruns lead to short interruptions of the audio signal. Buffer overruns lead to dropped samples. In this setup, buffer overruns do not immediately lead to dropped samples. Since writing to the audio device is blocking, buffer overruns delay reading data from the IEEE 1394 interface and, eventually, lead to a buffer overrun in the IEEE 1394 system with the consequence of dropped packets. If one of both interfaces (the IEEE 1394 or the audio subsystem) had buffers that automatically grow if necessary (which is not the case), the result would be even worse. The application would consume more and more memory and perhaps degrade performance of the system. Moreover, this would cause an increasing audio delay. To overcome these problems, a synchronisation mechanism is required. The main objective of the synchronisation is to keep the buffer usage in the audio subsystem within a reasonable range. It is not mandatory to precisely present each sample at the time designated by the timestamp in the IEEE 1394 packet because synchronous playback at several devices is not required, a constant delay and also a slight jitter are regarded tolerable.

The first approach was keeping the audio buffer filled about to the half. If the buffer usage drops below 25 %, the program will start doubling each sample until the buffer usage reaches 50 % again. If the buffer usage goes up over 75 %, the program will discard all newly received samples until the buffer usage reaches 50 %. It turned out that, on the hardware used, those synchronisation conditions trigger quite often, several times per hour (of course, depending on the audio hardware, some sound cards have faster, some have slower oscillators). And the artefacts are very annoying. In particular doubling samples, which, in fact, halves the audio frequencies and plays the signal one octave lower for a short time, is very annoying. A slightly better solution is to insert silence for that time period, but still this is very irritating. The time of interruption (a quarter of the audio buffer) is simply too much. A possibility to reduce the duration of the interruptions or dropped audio sequences is, of course, moving both limits for the buffer usage more to the middle, e. g. to 45 % and 55 %. However, some distance between these limits is needed because buffer usage will always vary over time. One reason is the scheduler interval of the operating system. While the operating system is executing other processes, the audio buffer is drained leading to lower buffer usages. On the other hand, as isochronous data is received from the IEEE 1394 API as chunks, it will fill the buffer to more than the target buffer usage. In the actual implementation, the IEEE 1394 API delivers chunks of between 250 and 1000 isochronous packets, so the audio buffer is always filled with at least slightly more than 30 ms at once. Of course, these timing parameters can be tweaked, but shorter intervals lead to a higher overhead and, therefore, a higher load of the system.

Algorithm 6.3 Calculation of audio deviation

```
presentationCycle ⇐ (receptionCycle & 0xfff0) {cut the 4 least significant bits of reception cycle}
presentationCycle ⇐ presentationCycle | (4 most significant bits of SYT) {combine it with the cycle info from SYT}
deviationTemp ⇐ presentationCycle − receptionCycle
if deviationTemp < −4 then {hold deviation in the interval [−4, 11]}
  presentationCycle ⇐ presentationCycle + 16
else if deviationTemp > 11 then
  presentationCycle ⇐ presentationCycle − 16
end if
presentationCycle ⇐ (presentationCycle + 8000) mod 8000 {compensate overflow}
get currentCycleTime {first measurement of current cycle time}
get audioDelay
maxDeviation ⇐ presentationCycle − currentCycleTime.cycleCount
get currentCycleTime {second measurement of current cycle time}
minDeviation ⇐ presentationCycle − currentCycleTime.cycleCount
{Compensate possible overflows, bring deviation into [−3999, 4000]}
if maxDeviation < −3999 then
  maxDeviation ⇐ maxDeviation + 8000
else if maxDeviation > 4000 then
  maxDeviation ⇐ maxDeviation − 8000
end if
if minDeviation < −3999 then
  minDeviation ⇐ minDeviation + 8000
else if minDeviation > 4000 then
  minDeviation ⇐ minDeviation − 8000
end if
maxDeviation ⇐ maxDeviation − audioDelay {subtract measured audio delay}
minDeviation ⇐ minDeviation − audioDelay {subtract measured audio delay}
maxDeviation ⇐ maxDeviation + AUDIODELAY {add constant audio delay}
minDeviation ⇐ minDeviation + AUDIODELAY {add constant audio delay}
if minDeviation > MAX_ALLOWED_DEVIATION then
  insert samples
else if maxDeviation < −MAX_ALLOWED_DEVIATION then
  drop samples
end if
```

Hence, a really satisfying solution cannot only look at the buffer usage. It has to use the timestamps inside the isochronous packets in order to find out whether a sample would be played in time or too early or to late, so that it can perform smaller adaptations at a higher rate. This calculation of the deviation of a sample from the correct time requires several steps, which are shown in Algorithm 6.3.

First, the timestamp in the `SYT` field of the CIP header contains the lower 16 bits of the IEEE 1394 at which a sample of this packet should be played back. These are the twelve bits of the cycle offset and the four least significant bits of the cycle count. This number runs over

every 16 cycles. Since processing of data usually happens more than 16 cycles after reception—as written above several hundred isochronous packets are processed at once—, it is necessary to reconstruct the missing 9 bits of the cycle count. As it is ensured (by choosing an appropriate IEEE 1394 buffer size of 1000 isochronous packets) that isochronous data is handled far less than one second after it has been received, it is not necessary to reconstruct the second count in order to get the correct presentation time. For each received isochronous packet, the IEEE 1394 API reports the cycle number at which it has been received. The algorithm assumes that the timestamp points either at most 4 cycles into the past with regard to the reception cycle or at most 11 cycles into the future. It would also have been possible to choose a symmetric interval (7 cycles into the past to 8 cycles into the future), but it is assumed that late packets are less probable than packets pointing more into the future. The four bits of the cycle count from the `SYT` field are combined with the nine most significant bits of the cycle number reported by the IEEE 1394 API. Then 16 is added or subtracted, if necessary, to bring the timestamp into the interval $[-4, 11]$ from the reception time. After this reconstruction, the correct cycle the `SYT` field refers to has been found. This timestamp does not refer to the presentation time of the first sample within the isochronous packet, but to a sample that could be figured out using the `SYT_INTERVAL` and the `DBC` (data block count) field of the CIP header. However, this difference is ignored because an accuracy of several isochronous cycles (i. e. several hundred microseconds) is sufficient. The timestamp is treated as presentation time of the first sample in the packet.

To find out, whether the samples in the isochronous packet would be played too late, too early, or just in time, two measurements are necessary: The current isochronous cycle time has to be measured and the delay caused by the audio buffer. Under ideal circumstances, these two measurements should be performed at exactly the same time. Unfortunately, the measurements can only be performed sequentially, and therefore, there is always some time between them. And this time can be significant if the scheduler of the operating system put the process asleep between these two measurements leading to an inaccuracy of the calculated deviation. To overcome this problem, the current IEEE 1394 cycle time is read twice, once before getting the audio delay and once afterwards. The calculation using the first value, which has been read before getting the audio delay, yields an upper limit for the deviation, the second calculation yields a lower limit. Both calculations must, of course, compensate overflows by adding or subtracting 8000 to the difference between the cycle number of the presentation timestamp and the current cycle time. It is assumed that this difference is always in the range $[-3999, 4000]$. After compensation of overflows, the measured delay caused by the audio buffer is subtracted from the deviation. This audio buffer delay determines how long it takes until a sample written to the audio subsystem would be played out. Furthermore, a constant delay is added in order to keep the audio buffer always filled to cope with latencies caused by the scheduler or the IEEE 1394 subsystem. This delay is set to 2400, which is equivalent to 0.3 s. This delay should, of course, always be less than the audio buffer size, this particular value was selected arbitrarily to fit to typical audio buffer sizes. So after this calculation, the exact value of the deviation is not known, but there is lower limit and an upper limit. A modification of the audio stream will only be performed, if the lower limit is higher than some threshold or the upper limit is below this threshold. So only in case the error is definitely known to be out of the allowed range, modifications will be performed. In case of uncertain deviation because of a too high measurement error, no modifications are performed. If the lower limit is too high, an appropriate number of samples will be inserted in order to compensate the deviation leading to a short period of silence. For this purpose, the first sample of the current isochronous packet is inserted several times. In case of the upper limit of the deviation being

too low, the corresponding number of samples will be dropped. The threshold, which designates the allowed maximum deviation, is defined as 100 isochronous cycles, which is equivalent to 12.5 ms. This means that the typical adjustments (inserted or dropped samples) also lie in this range. So there are quite many adaptations necessary, the interval between adaptation is in the range of minutes. They are clearly audible, but since they are small, they are not too annoying. This strategy of inserting or dropping samples is obviously not the best one. As already mentioned above, the best solution would be resampling the stream to slightly modify the sample rate in order to compensate differences between the sample frequency of the original stream and of the target device. Even for inserting or dropping samples, there would be better approaches leading to less disturbing artefacts by choosing a better time where samples are dropped and better sequences of samples to insert [107]. Since the computation of the deviation, in particular the necessary measurements, cause a significant load and the deviation usually increases or decreases slowly, this computation is not performed after each isochronous packet. The interval between calculation was set to 1000 isochronous cycles leading to 8 measurements and calculations per second. All these configuration parameters of the application can perhaps be optimised, but the values chosen are reasonable and lead to good results.

This gateway only implements the streaming part of a gateway. It does not provide the control part, not even the connection management procedures according to IEC 61883-1 are performed. In fact, it only allows listening to existing audio streams on the IEEE 1394 bus, which were set up by other devices or applications. Of course, it would be possible to integrate a control part into the software, obviously only for the IEEE 1394 part because the MOST part is just simulated by the Linux audio system. The program provides a simple text based user interface. This interface makes it possible that it can easily be integrated as streaming module into a bigger system. It would get commands for starting reception from a certain isochronous channel or stopping reception from the control part.

7 Conclusion

The aim of this work is to achieve convergence between various networks that are used within home environments, but also automotive environments. Real-time streaming of multimedia data imposes some requirements on the underlying networks and protocols. Different kinds of network used in home and automotive environments are evaluated with regard to real-time multimedia streaming. Various data formats and transport protocols that are used on different networks are described. Some fit well to the properties of one network, but are not appropriate or even not usable for other networks. For example, audio in the data format RealAudio contained in RTP packets is appropriate on IP based networks. However, on IEEE 1394, RTP packets can be transmitted (because IP can be used on top of IEEE 1394), but it is totally uncommon, and most devices—in particular consumer electronic devices—are not able to handle it. Similarly, the data format RealAudio, which is quite common on IP networks, is unknown to most typical devices on IEEE 1394. On the other hand, the most common data format for audio on IEEE 1394 are PCM coded audio samples—a data format that is problematic to transfer on some IP based networks due to bandwidth requirements. These transport protocols and data formats are usually used only within one network, but not across the borders between networks of different kind. The idea of the present work is that real-time multimedia data offered by the sources should be made available to as many sinks as possible, regardless which network they are connected to. Since different protocol suites are used on the various networks, the sources and sinks of multimedia streams cannot be coupled directly, gateways are necessary.

These gateways consist of two major parts, the streaming part and the control part, which interact with each other, but which are quite autonomous modules. As pointed out in Chapter 5 of this work, three different types of gateway to couple networks can be identified for the streaming part. This is based on the fact that transport of multimedia data can be split into two layers. There is a transport protocol, and within this transport protocol, multimedia data is contained in some data format.

- One type of gateway is the transport layer gateway that retransmits the stream in the same data format as it has been received, but within the other transport protocol that is used on the sink network. Besides the problem that this is not always possible because not every data format can be carried in every transport protocol, it has the major drawback, that the data format might be unusual on the sink network, with the consequence that it is not understood by many devices, and it might not well fit the features of the network, in particular the available bandwidth. The advantage of this kind of gateway is that it does not have to process the data format and, therefore, requires less computational resources and knowledge about the data format.

- Another one is the encapsulating gateway that transfers the transport protocol in capsules in the sink network. This gateway is primarily useful to tunnel a stream through a transit network. It is not suited to directly couple two networks because the devices on the sink network cannot handle the transport protocol of the source network contained in packets of the sink network.

- It turns out that only the most complex type of gateway, the data format layer gateway, which transforms the transport protocol as well as the data format, can really achieve the aim to make multimedia content available to a wide range of devices on the sink network. The other variants impose too many restrictions and enable only a subset of the sources and sink of real-time multimedia data within the home or car to take advantage of the services of devices in other networks. Only the data format layer gateway can ensure that both, transport protocol and data format, are understood by typical devices on the sink network.

In Chapter 5, furthermore, it is shown that one simple unidirectional gateway between two networks is not enough to achieve real convergence. Conversion in both directions is quite different, and therefore, a separate design of different gateways for both directions is an obvious solution. However, the modular design of a universal gateway presented in Chapter 5 of this thesis also allows integration of gateway functionality in both directions into one system by introducing appropriate input and output modules. Such gateway would even allow retransmission of a stream on the same network in another transport protocol or data format. This can be useful on networks where many different transport protocols and data formats exist, but the devices only support small subsets thereof. It solves the problem of two devices, a source and a sink, that do not support some common transport protocol or data format. In this case, the gateway as intermediate system can help the two devices on the same network that do not support a common transport protocol and data format to make use of the services offered by the other one. Certainly, this is just a workaround for the real problem of insufficient standardisation of the protocols and data formats or too many different protocols and data formats for the same purpose. The devices should negotiate some protocol that is supported by both, source and sink, and not require some intermediate system. It is a waste of bandwidth if the same content has to be transmitted twice on the same network at the same time in two different formats.

A module that only transforms real-time multimedia data streams between one network and another one alone does not provide a reasonable benefit. It must be coupled with some control module, which at least starts and stops multimedia streaming. As one example, on the IEEE 1394 side, AV/C or HAVi in connection with the connection management procedures defined in IEC 61883-1 is used for this purpose. In the IP world, RTSP is usually used for this purpose, maybe together with some higher layer protocol on top of it. As it is shown in Section 5.2, a control module on one side (e. g. AV/C and CMP on IEEE 1394) needs an interface to a corresponding module on the other side (e. g. RTSP on the IP network) to start or stop the stream on the other side, at the corresponding end devices, and perform configuration operations. So gateway functionality between these higher layer protocols is necessary. But it also needs an interface to the underlying real-time stream transformer. This is at least needed to start reception and transmission on the gateway. Hence, the streaming part and the control part can be seen as separate gateways on their own, working on different layers, but interaction between them is necessary.

This work shows that due to an abstract model of the streaming technologies used on different networks, it is possible to build a modular, universal gateway supporting the transfer of different kinds of real-time multimedia streams between two networks. The main point is that for the multimedia data in question, there always exists some plain unencoded data format, which all streams on the source network can be converted to and which all streams on the sink network can be converted from. Such universal data format can be PCM coded audio samples

or unencoded sequences of pictures, finally unencoded sequences of pixels. Input and output modules can be built on basis of these universal data formats. However, the control protocols, which also require to be coupled, do not allow such abstract view. They often use completely different and incompatible models of the network, also based on the fact that the networks themselves are based on very different concepts. Therefore, no way to design the control part of the gateway in such modular way that makes it extensible by introducing additional modules for different protocols is found. This would only be possible for small groups of protocols that share common concepts. But in general, it is necessary to develop different gateways, or different control parts, for different protocols. Of course, these different gateways might be integrated into one system, perhaps sharing the common streaming part, but they could also be implemented on completely different systems. If separate gateways are used for different control protocols, transport protocols, and data formats, it is clear that the number of necessary gateways will grow. Typically, there are not too many usual transport protocols on one kind of network, often there is only just one that is widely used, and also the number of different control protocols is not too high usually. The most variable part are the data formats, and they can be gotten under control by appropriate input and output modules for the streaming part.

Even though many gateways would be necessary in principle for the variety of different data types and formats available on the networks of interest, the architecture of a gateway presented in Chapter 5 demonstrates that a modular design makes it possible to integrate capabilities for different data formats and transport protocols into one single gateway. This design makes it easy to add support for additional data formats or transport protocols.

Another important result is that, in order to achieve convergence between various networks, it is inevitable that all devices strictly adhere to the appropriate standards. Standardisation per se is very important for interoperability. Even within one kind of network, it is important that the protocols and data formats are standardised and used by the devices. If interoperability between different networks should be achieved, standardisation is even more important. Devices relying on proprietary protocols or data formats will hardly be supported by gateways, unless these proprietary protocols are so widely used that they have become de facto standards. Even devices that use variants of widely used standards might be excluded from this convergence concept or can only expose a subset of their features to devices of other networks. It is very difficult for a generic gateway to support all protocols, data formats, and variants that could be used. In order to make the gateway not too complex, it will support the most commonly used protocols and data formats on the home or automotive networks. Exotic protocols and data formats will require separate gateways or special plugins or modules for the presented gateway.

While the main focus of this work is the streaming part of the gateway, for the control part the way for implementation is pointed out in Section 5.2, and some references to related work focusing on gateways for control purposes are given in Section 4.2. However, the actual implementations given here in Chapter 6 to prove the concept of the proposed gateway design all focus on the streaming part. The examples of implementations clearly show how the streaming part operates. The control part could definitely be a valuable subject for further investigation, in particular how to build it in a modular way, so that it can be extended for different control protocols with as little additional effort as possible. Some design choices for the control part are pointed out, for example when to start streaming on the source network.

One of the main issues for proper solutions to the problem of transmitting a real-time multimedia stream from one network onto another one is timing. It is important that the timing of the stream on the original network is preserved when retransmitting it on the target network.

All timing information from the original stream has to be transformed to the reference clock on the sink network. In the approach proposed in Section 5.1, the stream should be modified as little as possible. In particular, no samples or pictures should be added or dropped. Only for data format adaptation, e. g. scaling, the number of samples or pictures might be modified if necessary, but not for timing reasons, unless it is really necessary. The only scenario in which such modifications are necessary will arise if, on the sink network, there is some clock that the sample rate on the sink network has to be based on—by generating the sample clock from the reference clock frequency—and this clock cannot be synchronised to the sample rate on the source network. As one example for such kind of network, MOST is presented. In Section 6.3, one simple algorithm to slightly adapt the sample rate is demonstrated and implemented, but also more sophisticated methods, which would yield better quality, are lined out.

One problem concerning timing information is that not all timing information necessary for transmission of the multimedia stream is made available explicitly by packet headers. Instead, some information has to be extracted from the reception process. In particular, the exact reception times of single packets or the interarrival intervals have to be known. Unfortunately, this information often gets lost by the underlying hardware and software layers, which receive the packets, but hand it over to the upper layers in some aggregate form as big chunks of data without precise timing information or with incomplete timing information. While these bulk transfers are necessary for performance reasons, they introduce the difficulty that this timing information has to be reconstructed somehow. Section 6.1 extensively explains how such information can be reconstructed and gives many mathematical considerations on how certain parameters have to be chosen in order to fulfil the required tolerances. It turns out that, in particular in case of streams with very tight timing constraints like MPEG-2 transport streams, this is not quite easy. It requires assumptions about the reception process. In the shown examples, the incoming streams have constant sample rates or packet rates. Therefore, measurements of the average sample or packet arrival rates can be used to reconstruct the arrival times or, at least, the interarrival intervals. From interarrival intervals, the arrival times can be reconstructed with just a constant offset as error. And such constant offsets usually do not matter. Only in case inter-media synchronisation is necessary and these constant offsets are different for the involved streams, they have to be taken into account. In order to get good accuracy, long measurement times are required. The consequence is that there is a trade-off between long convergence times and delays on the one hand and sufficient accuracy on the other hand. However, it is shown that, even for MPEG-2 transport streams with very strict timing tolerances, a software based solution for transmission on IEEE 1394 with acceptable timing behaviour can be implemented. Nevertheless, although this solution can fulfil most timing constraints imposed by the according standards, there are some parameters that are not possible to be guaranteed to stay inside the imposed tolerances without requiring unacceptably long convergence times. Especially these timing issues are the reason that usually only hardware based solutions are regarded sufficient for such functionality. The main advantage of hardware based solutions is that it is much easier to design them in a way that the delay of the incoming data through the system is known very exactly and the tolerances can be set very precisely. Although hardware based solutions usually have better timing properties, this work demonstrates that proper software based solutions are possible as well if certain deviations from the very strict timing constraints were allowed. But it can also be seen that such software based solutions require some assumptions about the reception process. If such assumptions cannot be made, e. g. in case of a variable bit rate stream, it will be hard to build a software based solution. In contrast, in many cases, processing of the multimedia stream in software is necessary. In

particular, decoding or transcoding of multimedia data can often only be performed by the use of software decoders and encoders.

Besides the topic of coupling control protocols, which is addressed (Section 5.2), but which can be subject for further work, there is a second topic that needs more investigation. The area of content protection and digital rights management (DRM) is examined and possible solutions for adding support for it to multimedia streaming gateways are lined out in Section 5.3. However, there is not enough information available about the commercial systems currently in use to develop real solutions, most information is confidential and kept secret by the companies involved. Furthermore, all test streams used for the work on the present thesis are available freely and unencrypted. The implementations do not care about content protection issues and only operate on unencrypted multimedia streams. However, it is not yet clear to which extent such systems will be enforced in the future and whether it will be necessary to support them. This work lines out how such technologies could be integrated into a multimedia streaming gateway, but further investigation on this topic is necessary.

The implementations presented in Chapter 6 of this work demonstrate the feasibility of several aspects of multimedia streaming gateways. The audio gateway between IP and IEEE 1394 (Section 6.2) is a complete example of the streaming part of a data format layer gateway between these two networks. The control part is outside the main scope of the present work, but possible solutions are outlined. Although the implementation is still not as mature as it would be necessary for a product and has some weaknesses, it clearly shows that and how such gateway can work. For the streaming part of an audio gateway between IEEE 1394 and MOST, the MOST side is emulated on a sound card (Section 6.3). Although there is no transmission over a real sink network, the sound card represents the synchronous behaviour of the MOST bus well. On the IEEE 1394 side, the input module for audio from IEEE 1394 with perfect reconstruction of all necessary timing information works very well. It is a transport layer gateway that can be easily extended to a data format layer gateway. The gateway between DVB and IEEE 1394 (Section 6.1) is an example of the streaming part of a transport layer gateway. It has no typical home network on the source side, but such gateway is a very interesting application from the user's point of view because TV reception systems are very common and distributing TV content inside the home increases the benefit of these systems. The input module from the DVB side makes timing issues very clear and shows how missing information can be reconstructed. The output modules demonstrates how MPEG-2 transport streams can be packetised on IEEE 1394 with the timing enforced by an external source system. This gateway also demonstrates adaptations to the MPEG-2 transport stream by filtering packets and transforming the full MPEG-2 transport stream into a partial MPEG-2 transport stream. This helps reducing the required bandwidth on the sink network by eliminating information that is not needed at the sink.

Altogether, the implementations prove that convergence between networks in home and automotive environments can be achieved. Although all these gateways need further work to improve functionality and add missing parts, all components necessary for multimedia streaming gateways are discussed in this thesis and can be put together in order to create fully functional gateways.

Abbreviations

AAC	Advanced Audio Coding
ACELP	Algebraic Code Excited Linear Prediction
ADC	Analogue to Digital Converter
ADM	Adaptive Delta Modulation
ADPCM	Adaptive DPCM
ALSA	Advanced Linux Sound Architecture
ANSI	American National Standards Institute
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
ATRAC	Adaptive Transform Acoustic Coding
ATSC	Advanced Television Systems Committee
A/V	Audio/Video
AVC	Advanced Video Coding
AV/C	Audio/Video Control
AVI	Audio Video Interleave
BAV	Base A/V Device
CA	Conditional Access
CAN	Controller Area Network
CCIR	Consultative Committee for International Radio
CCP	Customer Convenience Port
CD	Compact Disc
CD-DA	Compact Disc – Digital Audio
CELP	Code Excited Linear Prediction
CIF	Common Interchange Format
CIP	Common Isochronous Packet
CMP	Connection Management Procedures
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check

CRT	Cathode Ray Tube
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DAC	Digital to Analogue Converter
DCM	Device Control Module
DCT	Discrete Cosine Transformation
DIF	Digital Interface Format
DIT	Discontinuity Information Table
DM	Delta Modulation
DMA	Direct Memory Access
DPCM	Differential PCM
DRM	Digital Rights Management
DSS	Digital Satellite System
DSSS	Direct Sequence Spread Spectrum
DTCP	Digital Transmission Content Protection
DTLA	Digital Transmission Licensing Administrator
DTS	Decoding Timestamp
DV	Digital Video
DVB	Digital Video Broadcasting
DVB-S	DVB Satellite
DVD	Digital Versatile Disc
EIB	European Installation Bus
EIT	Event Information Table
EMC	Electromagnetic Compatibility
FAV	Full A/V Device
FCM	Functional Component Module
FEC	Forward Error Correction
FHSS	Frequency Hopping Spread Spectrum
FLAC	Free Lossless Audio Codec
FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
GOP	Group of Pictures

GSM	Global System for Mobile Communications
GUID	Global Unique Identifier
HAVi	Home Audio/Video Interoperability
HDTV	High Definition Television
HiFi	High Fidelity
HPCF	Hard Polymer Clad Fibre
HTTP	Hypertext Transfer Protocol
IAV	Intermediate A/V Device
ICMP	Internet Control Message Protocol
IDB	Intelligent Data Bus
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical an Electronics Engineers, Inc.
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
I^2C	Inter-Integrated Circuit
IP	Internet Protocol
iPCR	Input Plug Control Register
ISDN	Integrated Services Digital Network
ISM	Industrial, Scientific, Medical
ISO	International Organization for Standardization
ITU	International Telecommunication Union
ITU-T	ITU Telecommunications Bureau
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
LAV	Intermediate A/V Device
LFE	Low Frequency Enhancement
MAN	Metropolitan Area Network
M-JPEG	Motion-JPEG
MLD	Multicast Listener Discovery
MMF	Multimode Fibre
MMS	Microsoft Media Server
MOST	Media Oriented Systems Transport
MPEG	Moving Picture Experts Group

NIT	Network Information Table
NTP	Network Time Protocol
NTSC	National Television System Committee
OFDM	Orthogonal Frequency Division Multiplexing
OHCI	Open Host Controller Interface
oPCR	Output Plug Control Register
OSI	Open Systems Interconnection
PAL	Phase Alternating Line *or* Protocol Adaptation Layer
PAT	Program Association Table
PC	Personal Computer
PCI	Peripheral Component Interconnect
PCM	Pulse Code Modulation
PCR	Program Clock Reference *or* Plug Control Register
PES	Packetised Elementary Stream
PID	Packet Identifier
PMT	Program Map Table
POF	Plastic Optical Fibre
POSIX	Portable Operating System Interface
PS	Program Stream
PSI	Program Specific Information
PTS	Presentation Timestamp
QCIF	Quarter CIF
QoS	Quality of Service
RDT	Real Delivery Transport
RFC	Request for Comments
RSVP	Resource Reservation Protocol
RTCP	RTP Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
SCR	System Clock Reference
SCSI	Small Computer System Interface
SD	Standard Definition
SECAM	Séquentiel Couleur à Mémoire, *French for* Sequential Colour with Memory

SI	Service Information
SID	Source Node ID
SIF	Source Input Format *or* Standard Interchange Format
SIT	Selection Information Table
SNR	Signal to Noise Ratio
SR	Sender Report
SRM	System Renewability Message
ST-II	Internet Stream Protocol, Version 2
STP	Shielded Twisted Pair
SYT	Synchronisation Timestamp
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TS	Transport Stream
TV	Television
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
UPnP	Universal Plug and Play
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTP	Unshielded Twisted Pair
VCR	Video Cassette Recorder
VHS	Video Home System
WAN	Wide Area Network
WLAN	Wireless LAN
WMA	Windows Media Audio

Bibliography

[1] InHoMNet Consortium. In-Home High Speed Multimedia Network (InHoMNet) based on IEEE 1394 – Final Public Report. Report IST-1999-10622, IST, February 2002.

[2] Brendan Roycroft, Brian Corbett, Carmel Kelleher, John Lambkin, Baudouin Bareel, Jacques Goudeau, and Peter Skiczuk. The HomePlanet project: a HAVi multi-media network over POF. In John G. McInerney, Gerard Farrell, David M. Denieffe, Liam P. Barry, Harold S. Gamble, Padraig J. Hughes, and Alan Moore, editors, *Opto-Ireland 2005: Optoelectronics, Photonic Devices, and Optical Networks*, volume 5825, pages 574–584, Dublin, Ireland, 2005. SPIE.

[3] Ralf Steinmetz. *Multimedia-Technologie – Grundlagen, Komponenten und Systeme*. Springer, Berlin, Heidelberg, Germany, second edition, 1999. ISBN 3-540-62060-5.

[4] François Fluckiger. *Multimedia im Netz*. Prentice Hall, München, Germany, first edition, 1996. ISBN 3-8272-9522-X.

[5] Dietmar Dietrich, Dietmar Loy, and Hans-Jörg Schweinzer, editors. *LON-Technologie. Verteilte Systeme in der Anwendung*. Hüthig, second edition, 1999. ISBN 3-7785-2770-3.

[6] Dietmar Dietrich, Wolfgang Kastner, and Thilo Sauter, editors. *EIB Gebäudebussystem*. Hüthig, 2000. ISBN 3-7785-2795-9.

[7] Hans-Jörg Schweinzer. LonVoice – Transmitting Control Data and Voice by using the LonTalk Protocol. In Dietmar Dietrich and Herbert Schweinzer, editors, *Feldbustechnik in Forschung, Entwicklung und Anwendung*, pages 167–175. Springer, 1997. ISBN 3-211-83062-6.

[8] Don Anderson. *FireWire System Architecture: IEEE 1394a*. Addison-Wesley, Reading, MA, USA, second edition, 1999. ISBN 0-201-48535-4.

[9] Andrew S. Tanenbaum. *Computer Networks*. Paerson Education International, Upper Saddle River, New Jersey, USA, fourth edition, 2003. ISBN 0-13-038488-7.

[10] Detlef Vieweg. IDB 1394 oder MOST. *HANSER automotive*, 2003(5):47–49, 2003. ISSN 1860-5699.

[11] Paul Polishuk. Automotive Industry in Europe Takes the Lead in the Introduction of Optical Data Buses. *Wiring Harness News*, November/December 2001.

[12] Aart van Halteren, Leonard Franken, Dave de Vries, Ing Widya, Gloria Túquerres, Johan Pouwelse, and Phil Copeland. QoS architectures and mechanisms. Deliverable 3.1.1, AMIDST project, January 1999. Reference: AMIDST/WP3/N005/V09.

[13] ISO/IEC. Information technology – Open Distributed Processing – Reference Model: Foundations. Standard ISO/IEC 10746-2, ISO/IEC, Geneva, Switzerland, 1996. also: ITU-T X.902.

[14] ITU-T. Terms and Definitions related to Quality of Service and Network Performance including Dependability. Recommendation E.800, ITU-T, Geneva, Switzerland, August 1994.

[15] L. Mejlbro. QOSMIC-deliverable D1.3C: QoS and Performance Relationships. Deliverable QOSMIC R1082, RACE, 1992.

[16] Andreas Vogel, Brigitte Kerhervé, Gregor von Bochmann, and Jan Gecsei. Distributed Multimedia and QOS: A Survey. *IEEE Multimedia*, 2(2):10–19, Summer 1995. ISSN 1070-986X. doi: 10.1109/93.388195.

[17] Hubert Zimmermann. OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980. ISSN 0096-2244.

[18] John D. Day and Hubert Zimmermann. The OSI Reference Model. *Proceedings of the IEEE*, 71(12):1334–1340, December 1983. ISSN 0018-9219.

[19] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Standard IEEE 1394-1995, IEEE Computer Society, New York, USA, 1995.

[20] Ralph Wittmann and Martina Zitterbart. *Multicast Communication – Protocols and Applications*. Academic Press, San Diego, USA, 2001. ISBN 1-55860-645-9.

[21] ITU-T. Pulse code modulation (PCM) of voice frequencies. Recommendation G.711, ITU-T, Geneva, Switzerland, November 1988.

[22] Claude Elwood Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[23] Mat Hans and Ronald W. Schafer. Lossless Compression of Digital Audio. Technical Report HPL-1999-144, HP Laboratories Palo Alto, 1999.

[24] John Watkinson. *The MPEG Handbook – MPEG-1, MPEG-2, MPEG-4*. Focal Press, first edition, 2001. ISBN 0-240-51656-7.

[25] ISO/IEC. Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio. International Standard ISO/IEC 11172-3, First edition, ISO/IEC, 1993.

[26] ISO/IEC. Information technology – Generic coding of moving pictures and associated audio information – Part 3: Audio. International Standard ISO/IEC 13818-3, First edition, ISO/IEC, 1996.

[27] ISO/IEC. Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC). International Standard ISO/IEC 13818-7, First edition, ISO/IEC, 2003.

[28] ISO/IEC. Information technology – Coding of audio-visual objects – Part 3: Audio. International Standard ISO/IEC 14496-3, Second edition, ISO/IEC, 2001.

[29] Gerald Himmelein. Aus halb mach ganz – Deinterlacing: Wie DVD-Player Bilder wiedergeben. *c't magazin für computer technik*, 2004(25):204–206, November 2004. ISSN 0724-8679.

[30] ITU-R. Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. Recommendation BT.601-5, ITU-R, Geneva, Switzerland, October 1995.

[31] ISO/IEC. Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 2: Video. International Standard ISO/IEC 11172-2, First edition, ISO/IEC, 1993.

[32] ITU-T. Video codec for audiovisual services at p x 64 kbit/s. Recommendation H.261, ITU-T, Geneva, Switzerland, March 1993.

[33] ISO/IEC. Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines. International Standard ISO/IEC 10918-1, First edition, ISO/IEC, 1994.

[34] IEC. Recording – Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems) – Part 2: SD format for 525-60 and 625-50 systems. Standard IEC 61834-2, First edition, IEC, Geneva, Switzerland, August 1998.

[35] IEC. Recording – Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems) – Part 3: HD format for 1125-60 and 1250-50 systems. Standard IEC 61834-3, First edition, IEC, Geneva, Switzerland, November 1999.

[36] IEC. Recording – Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems) – Part 6: SDL format. Standard IEC 61834-6, First edition, IEC, Geneva, Switzerland, August 2000.

[37] IEC. Consumer audio/video equipment – Digital Interface – Part 2: SD-DVCR data transmission. Standard IEC 61883-2, First edition, IEC, Geneva, Switzerland, February 1998.

[38] IEC. Consumer audio/video equipment – Digital Interface – Part 3: HD-DVCR data transmission. Standard IEC 61883-3, First edition, IEC, Geneva, Switzerland, February 1998.

[39] IEC. Consumer audio/video equipment – Digital Interface – Part 5: SDL-DVCR data transmission. Standard IEC 61883-5, First edition, IEC, Geneva, Switzerland, February 1998.

[40] ISO/IEC. Information technology – Generic coding of moving pictures and associated audio information: Video. International Standard ISO/IEC 13818-2, First edition, ISO/IEC, 1996.

[41] ETSI. Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems. European Standard EN 300 468, version 1.6.1, ETSI, November 2004.

[42] ISO/IEC. Information technology – Coding of audio-visual objects – Part 2: Visual. International Standard ISO/IEC 14496-2, Third edition, ISO/IEC, 2004.

[43] ISO/IEC. Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding. International Standard ISO/IEC 14496-10, Second edition, ISO/IEC, 2004.

[44] Dirk Knopp. Effizientissimo – Die wichtigsten Unterschiede zwischen MPEG-4 ASP, H264/AVC und VC-1. *c't magazin für computer technik*, 2005(10):158–160, May 2005. ISSN 0724-8679.

[45] ITU-T. Information technology – generic coding of moving pictures and associated audio information: Video. Recommendation H.262, ITU-T, Geneva, Switzerland, February 2000.

[46] ITU-T. Video coding for low bit rate communication. Recommendation H.263, ITU-T, Geneva, Switzerland, January 2005.

[47] ITU-T. Advanced video coding for generic audiovisual services. Recommendation H.264, ITU-T, Geneva, Switzerland, May 2003.

[48] ISO/IEC. Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 1: Systems. International Standard ISO/IEC 11172-1, First edition, ISO/IEC, 1993.

[49] ISO/IEC. Information technology – Generic coding of moving pictures and associated audio information: Systems. International Standard ISO/IEC 13818-1, Second edition, ISO/IEC, 2000.

[50] Albrecht Ziemer. *Digitales Fernsehen – Eine neue Dimension der Medienvielfalt.* Hüthig, Heidelberg, Germany, second edition, 1997. ISBN 3-7785-2559-X.

[51] Advanced Television Systems Committee. ATSC Digital Television Standard, Revision E with Amendment No. 1. Standard A/53E, Advanced Television Systems Committee, Washington, D. C., USA, December 2005.

[52] *QuickTime File Format.* Apple Computer, Inc., March 2001.

[53] Tobias Künkel. *Streaming Media – Technologien, Standards, Anwendungen.* Addison Wesley, 2001. ISBN 3-8273-1798-3.

[54] Peter Skiczuk. *Network Protocol Architecture for Home Access Points.* Ph. D. dissertation, Vienna University of Technology, 2002.

[55] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus – Amendment 1. Standard IEEE 1394a-2000, IEEE Computer Society, New York, USA, March 2000.

[56] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus – Amendment 2. Standard IEEE 1394b-2002, IEEE Computer Society, New York, USA, December 2002.

[57] 1394 Trade Association. IDB-1394 Automotive Specification 1.0. TA Document 2001018, 1394 Trade Association and IDB-Forum, Grapevine, Texas, USA, March 2003.

[58] IEC. Consumer audio/video equipment – Digital Interface – Part 1: General. Standard IEC 61883-1, First edition, IEC, Geneva, Switzerland, February 1998.

[59] 1394 Trade Association. *AV/C Digital Interface Command Set, General Specification, Version 4.2*. 1394 Trade Association, Southlake, Texas, USA, September 2004. TA Document number 2004006.

[60] HAVi Inc. *Specification of the Home Audio/Video Interoperability (HAVi) Architecture*. HAVi, Inc, 1.1 edition, May 2001.

[61] Manfred Weihs and Michael Ziehensack. Convergence between IEEE 1394 and IP for Real-time A/V Transmission. In Dietmar Dietrich, Peter Neumann, and Jean-Pierre Thomesse, editors, *Fieldbus Systems and their Applications 2003*, pages 299–305. IFAC, Elsevier, 2003. ISBN 0-08-044247-1.

[62] IEC. Consumer audio/video equipment – Digital Interface – Part 6: Audio and music data transmission protocol. Standard IEC 61883-6, First edition, IEC, Geneva, Switzerland, October 2002.

[63] IEC. Consumer audio/video equipment – Digital Interface – Part 4: MPEG2-TS data transmission. Standard IEC 61883-4, First Edition, IEC, Geneva, Switzerland, February 1998.

[64] DVD Forum. *Guideline of Transmission and Control for DVD-Video/Audio through IEEE 1394 Bus, Version 1.0*. DVD Forum, September 2002.

[65] IEC. Consumer audio/video equipment – Digital Interface – Part 7: Transmission of Rec. ITU-R BO.1294 System B Transport 1.0. Standard IEC 61883-7, First edition, IEC, Geneva, Switzerland, July 2001.

[66] IEEE Computer Society. IEEE Standard for High Performance Serial Bus Bridges. Standard IEEE 1394.1-2004, IEEE Computer Society, New York, USA, December 2004.

[67] P. Johansson. IPv4 over IEEE 1394. RFC 2734, Internet Engineering Task Force, December 1999.

[68] K. Fujisawa and A. Onoe. Transmission of IPv6 packets over IEEE 1394 networks. RFC 3146, Internet Engineering Task Force, October 2001.

[69] IEEE Computer Society. IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. Standard IEEE 802.3-2002, IEEE Computer Society, New York, USA, 2002.

[70] J. B. Postel. Internet protocol. RFC 791, Internet Engineering Task Force, September 1981.

[71] B. Cain, S. E. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet group management protocol, version 3. RFC 3376, Internet Engineering Task Force, October 2002.

[72] R. Braden an L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 functional specification. RFC 2205, Internet Engineering Task Force, September 1997.

[73] Henning Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a transport protocol for real-time applications. RFC 1889, Internet Engineering Task Force, January 1996.

[74] Henning Schulzrinne, Asha Rao, and R. Lanphier. Real time streaming protocol (RTSP). RFC 2326, Internet Engineering Task Force, April 1998.

[75] Sun microsystems. *Jini Architecture Specification Version 2.0Beta*. Sun Microsystems, Inc., February 2003.

[76] W. Keith Edwards. *Core Jini*. Prentice Hall, 1999. ISBN 3-8272-9592-0.

[77] UPnP Forum. *UPnP Device Architecture*. UPnP Forum, 1.0 edition, June 2000.

[78] Gerard O'Driscoll. *The Essential Guide to Home Networking Technologies*. Prentice Hall PTR, 2001. ISBN 0-13-019846-3.

[79] Henning Schulzrinne. RTP profile for audio and video conferences with minimal control. RFC 1890, Internet Engineering Task Force, January 1996.

[80] K. Kobayashi, A. Ogawa, S. Casner, and C. Bormann. RTP payload format for 12-bit DAT audio and 20- and 24-bit linear sampled audio. RFC 3190, Internet Engineering Task Force, January 2002.

[81] D. Hoffman, G. Fernando, V. Goyal, and M. Reha Civanlar. RTP payload format for MPEG1/MPEG2 video. RFC 2250, Internet Engineering Task Force, January 1998.

[82] K. Kobayashi, A. Ogawa, S. Casner, and C. Bormann. RTP payload format for DV (IEC 61834) video. RFC 3189, Internet Engineering Task Force, January 2002.

[83] R. Hinden. Proposed TLA and NLA assignment rule. RFC 2450, Internet Engineering Task Force, December 1998.

[84] Holger Fahner, Peter Feil, and Tanja Zseby. *MBone – Aufbau und Einsatz von IP-Multicast-Netzen*. dpunkt, Heidelberg, Germany, 2001. ISBN 3-920993-99-3.

[85] Ed. R. Vida and Ed. L. Costa. Multicast listener discovery version 2 (mldv2) for IPv6. RFC 3810, Internet Engineering Task Force, June 2004.

[86] MOST Cooperation. MOST Specification. Specification Revision 2.3, MOST Cooperation, Karlsruhe, Germany, August 2004.

[87] MOST Cooperation. MAMAC Specification. Specification Revision 1.1, MOST Cooperation, Karlsruhe, Germany, December 2003.

[88] Mike Sexton and Andy Reid. *ATM, SDH and SONET*. Artech House, Inc, 1997. ISBN 0-89006-578-0.

[89] IEEE Computer Society. IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs). Standard IEEE 802.15.1-2002, IEEE Computer Society, New York, USA, 2002.

[90] IEEE Computer Society. IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). Standard IEEE 802.15.4-2003, IEEE Computer Society, New York, USA, 2003.

[91] ZigBee Alliance. ZigBee Specification, Version 1.0. ZigBee Document 053474r06, ZigBee Standards Organization, December 2004.

[92] IEEE Computer Society. Information technology—Telecommunications and information exchange between systems – Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. Standard IEEE 802.11-1997, IEEE Computer Society, New York, USA, 1997.

[93] IEEE Computer Society. Supplement to IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz band. Standard IEEE 802.11a-1999, IEEE Computer Society, New York, USA, 1999.

[94] IEEE Computer Society. Supplement to IEEE Standard For Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Higher-speed Physical Layer Extension in the 2.4 GHz Band. Standard IEEE 802.11b-1999, IEEE Computer Society, New York, USA, 1999.

[95] IEEE Computer Society. IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications—Amendment 4: Further Higher Data Rate Extension in 2.4 GHz Band. Standard IEEE 802.11g-2003, IEEE Computer Society, New York, USA, November 2003.

[96] ETSI. Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; System Overview. Technical Report TR 101 683, version 1.1.1, ETSI, February 2000.

[97] ETSI. Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer. Technical Specification TS 101 475, version 1.3.1, ETSI, December 2001.

[98] Compaq, Intel, Microsoft, and NEC. *Universal Serial Bus Specification, Revision 1.1.* USB Implementers Forum, Inc., September 1998.

[99] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips. *Universal Serial Bus Specification, Revision 2.0.* USB Implementers Forum, Inc., April 2000.

[100] Norbert Stampfl. *IEEE 1394 as Control Network and High-Speed Backbone for Home and Industrial Automation.* Ph. D. dissertation, Vienna University of Technology, 2001.

[101] S. R. Subramanya and Byung K. Yi. Digital rights management. *IEEE Potentials*, 25 (2):31–34, March/April 2006. ISSN 0278-6648. doi: 10.1109/MP.2006.1649008.

[102] Willem Jonker and Jean-Paul Linnartz. Digital Rights Management in Consumer Electronics Products. *IEEE Signal Processing Magazine*, 21(2):82–91, March 2004. ISSN 1053-5888. doi: 10.1109/MSP.2004.1276116.

[103] Ltd. Hitachi, Intel Corporation, Ltd. Matsushita Electric Industrial Co., Sony Corporation, and Toshiba Corporation. Digital Transmission Content Protection Specification Volume 1 (Informational Version). Specification Revision 1.4, 5C, February 2005.

[104] ETSI. Digital Video Broadcasting (DVB); Support for use of scrambling and Conditional Access (CA) within digital broadcasting systems. ETSI Technical Report ETR 289, ETSI, October 1996.

[105] Yusei Nishimoto, Akitsugu Baba, Tatsuya Kurioka, and Seiichi Nambaa. A Digital Rights Management System for Digital Broadcasting Based on Home Servers. *IEEE Transactions on Broadcasting*, 52(2):167–172, June 2006. ISSN 0018-9316. doi: 10.1109/TBC.2006.875650.

[106] Gerold Blakowski and Ralf Steinmetz. A Media Synchronization Survey: Reference Model, Specification, and Case Studies. *IEEE Journal on Selected Areas in Communications*, 14(1):5–35, January 1996. ISSN 0733-8716. doi: 10.1109/49.481691.

[107] Orion Hodson, Colin Perkins, and Vicky Hardman. Skew Detection and Compensatin for Internet Audio Applications. In *2000 IEEE International Conference on Multimedia and Expo (ICME 2000)*, volume 3, pages 1687–1690, New York, USA, 2000. doi: 10.1109/ICME.2000.871096.

[108] Akimichi Ogawa, Katsushi Kobayashi, Kazunori Sugiura, Osamu Nakamura, and Jun Murai. Design and Implementation of DV based video over Internet. In *Internet Workshop 1999, IWS 99*, pages 255–260, Osaka, Japan, February 1999. IEEE. ISBN 0-7803-5925-9. doi: 10.1109/IWS.1999.811022.

[109] Akimichi Ogawa, Katsushi Kobayashi, Kazunori Sugiura, Osamu Nakamura, and Jun Murai. Design and Implementation of DV based video over RTP. In *Packet Video 2000*, number 31 in Proceedings, Forte Village Resort (Ca), Italy, May 2000. University of Cagliari.

[110] Takeshi Saito, Ichiro Tomoda, Yoshiaki Takabatake, Junko Ami, and Keiichi Teramoto. Home Gateway Architecture and its Implementation. *IEEE Transactions on Consumer Electronics*, 46(4):1161–1166, November 2000. ISSN 0098-3063. doi: 10.1109/30.920474.

[111] Takeshi Saito, Ichiro Tomoda, Yoshiaki Takabatake, Keiichi Teramoto, and Kensaku Fumimoto. Wireless Gateway for Wireless Home AV Network and its Implementation. *IEEE Transactions on Consumer Electronics*, 47(3):496–501, August 2001. ISSN 0098-3063. doi: 10.1109/30.964138.

[112] Takeshi Saito, Ichiro Tomoda, Yoshiaki Takabatake, Keiichi Teramoto, and Kensaku Fujimoto. Gateway Technologies for Home Network and Their Implementations. In *2001 International Conference on Distributed Computing Systems Workshop*, pages 175–180, Mesa AZ, USA, April 2001. IEEE. doi: 10.1109/CDCS.2001.918702.

[113] Roger Zimmermann, Kun Fu, Cyrus Shahabi, Didi Shu-Yuen Yao, and Hong Zhu. Yima: Design and Evaluation of a Streaming Media System for Residential Broadband Services. In Willem Jonker, editor, *Databases in Telecommunications II : VLDB 2001 International Workshop, DBTel 2001 Rome, Italy, September 10, 2001. Proceedings*, number 2209 in LNCS, pages 116–125, Rome, Italy, September 2001. VLDB Endowment, Springer-Verlag. ISBN 3-540-42623-X.

[114] Steve Bard. Wireless Convergence of PC and Consumer Electronics in the e-Home. *Intel Technology Journal*, 5(2), May 2001. ISSN 1535-766X.

[115] 1394 Trade Association Wireless Working Group. Protocol Adaptation Layer (PAL) for IEEE 1394 over IEEE 802.15.3. TA Document 2003010, 1394 Trade Association, Grapevine, Texas, USA, May 2004.

[116] IEEE Computer Society. IEEE Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements—Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPANs). Standard IEEE 802.15.5-2003, IEEE Computer Society, New York, USA, 2003.

[117] ETSI. Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Packet based Convergence Layer; Part 3: IEEE 1394 Service Specific Convergence Sublayer (SSCS). Technical Specification TS 101 493-3, version 1.2.1, ETSI, December 2001.

[118] ETSI. Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Packet based Convergence Layer; Part 4: IEEE 1394 Bridge Specific Functions sub-layer for restricted topology. Technical Specification TS 101 493-4, version 1.1.1, ETSI, July 2001.

[119] Michael Ziehensack. *Dynamische Kopplung von Gerätenetzwerken im Heimbereich*. Ph. D. dissertation, Vienna University of Technology, 2004.

[120] Manfred Weihs. Multimedia Streaming in Home Environments. In Pascal Lorenz and Petre Dini, editors, *Networking – ICN 2005: 4th International Conference on Networking, Reunion Island, France, April 17-21, 2005, Proceedings, Part I*, number 3420 in LNCS, pages 873–881, Reunion Island, France, April 2005. IEEE, IEE, IARIA, Springer-Verlag. ISBN 3-540-25339-4. doi: 10.1007/b107117.

[121] Anthony Vetro, Charilaos Christopoulos, and Huifang Sun. Video transcoding architectures and techniques: An overview. *IEEE Signal Processing Magazine*, 20(2):18–29, March 2003. ISSN 1053-5888. doi: 10.1109/MSP.2003.1184336.

[122] Jun Xin, Chia-Wen Lin, and Ming-Ting Sun. Digital video transcoding. *Proceedings of the IEEE*, 93(1):84–97, January 2005. ISSN 0018-9219. doi: 10.1109/JPROC.2004.839620.

[123] Manfred Weihs. Design Issues for Multimedia Streaming Gateways. In Pascal Lorenz, Petre Dini, Damien Magoni, and Abdelhamid Mellouk, editors, *International Conference on Networking (ICN 2006)*, Morne, Mauritius, April 2006. IEEE, IARIA, IEEE Computer Society. ISBN 0-7695-2570-9. doi: 10.1109/ICNICONSMCL.2006.74.

[124] Manfred Weihs. DVB Software-Dekodierung unter Linux. Master's thesis, Vienna University of Technology, 2000.

[125] Ltd. Hitachi, Intel Corporation, Ltd. Matsushita Electric Industrial Co., Sony Corporation, and Toshiba Corporation. 5C Digital Transmission Content Protection White Paper. White Paper Revision 1.0, 5C, July 1998.

[126] Elmar Nadschläger. Übertragung von DVB-Daten auf ein IEEE-1394-Netzwerk. Master's thesis, Vienna University of Technology, 2005.

[127] *Philips Semiconductors PDI1394L11/P11 Firewire Reference Design Kit User's Manual*. Philips Semicoductors, 1.2 edition, 1998.

[128] *Philips Semiconductors PDI1394L11/P11 Firewire Reference Design Kit Hardware Reference Manual*. Philips Semicoductors, 1.0 edition, 1997.

[129] IEEE Computer Society. Standard for Information Technology—Portable Operating System Interface (POSIX). System Interfaces. Standard IEEE Std 1003.1, 2004 Edition, IEEE Computer Society, New York, USA, 2004.

[130] *Unibrain FireAPI Documentation. 1394 Class Driver User Mode Interface*. Unibrain S.A, 2000.

[131] Christian Tögel. Weiterentwicklung des Abstraction Layers für IEEE 1394 und Implementierung für Linux. Master's thesis, Vienna University of Technology, 2002.

[132] ISO/IEC. Programming languages—C. International Standard ISO/IEC 9899:1999, Second edition, ISO/IEC, December 1999.

[133] Inc. Apple Computer, Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, National Semiconductor Corporation, Inc. Sun Microsystems, and Inc Texas Instruments. 1394 open host controller interface specification. Specification Release 1.1, Promoters of the 1394 Open HCI, January 2000.

[134] ETSI. Digital Video Broadcasting (DVB); Implementation guidelines for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream. Technical Specification TS 101 154, version 1.6.1, ETSI, January 2005.

[135] Mario Ruthmair. Gateway zur Übertragung von Audiodaten von einem RTSP-Server in ein IEEE-1394-Netzwerk. Master's thesis, Vienna University of Technology, 2006.

[136] D. L. Mills. Network time protocol (version 3) specification, implementation. RFC 1305, Internet Engineering Task Force, March 1992.

[137] Ronald E. Crochiere and Lawrence R. Rabiner. *Multirate Digital Signal Processing*. Prentice-Hall, 1983. ISBN 0-13-605162-6.

[138] Jaroslav Kysela, Abramo Bagnara, Takashi Iwai, and Frank van de Pol. *Alsa Project— The C Library Refrence*, 2006.

[139] Jeff Tranter and Hannu Savolainen. *Open Sound System – Programmer's Guide*. 4Front Technologies, Culver City, USA, 1.1 edition, November 2000.

Lebenslauf

Persönliche Daten

Name	Dipl.-Ing. Manfred Weihs
Anschrift	Babogasse 3
	2020 Hollabrunn
Geboren am	30. Dezember 1975
	in Hollabrunn
Familienstand	ledig
Staatsbürgerschaft	Österreich

Berufstätigkeit

seit Dezember 2000	Anstellung als Projektassistent am Institut für Computertechnik der TU Wien, Tätigkeitsbereiche: IEEE 1394 und Multimedia, Energie und Kommunikation
November 2001	Gründung der Firma dmn Software-Entwicklung GmbH als einer von fünf Gesellschaftern
2001 – 2004	Software-Entwicklung für die Firma dmn Software-Entwicklung GmbH
Oktober 1995 – Mai 1996	Präsenzdienst beim Österreichischen Bundesheer

Ausbildung

2001 – 2006	Doktoratsstudium der technischen Wissenschaften an der Fakultät für Elektrotechnik und Informationstechnik der TU Wien
1995 – 2000	Diplomstudium der Elektrotechnik an der TU Wien, Studienzweig Computertechnik, Studienabschluss im Oktober 2000 mit ausgezeichnetem Erfolg
1986 – 1994	Bundesgymnasium und Bundesrealgymnasium Hollabrunn, Matura mit ausgezeichnetem Erfolg
1982 – 1986	Volksschule Koliskoplatz in Hollabrunn