Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 103
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 103
Espoo 2006 HUT-TCS-A103

A WIRELESS MULTICAST DELIVERY ARCHITECTURE FOR MOBILE TERMINALS

Janne Lundberg

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 103
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 103
Espoo 2006 HUT-TCS-A103

A WIRELESS MULTICAST DELIVERY ARCHITECTURE FOR MOBILE TERMINALS

Janne Lundberg

Dissertation for the degree of Doctor of Science in Technology to be presented with due permission of the Department of Computer Science and Engineering, for public examination and debate in Auditorium T2 at Helsinki University of Technology (Espoo, Finland) on the 15th of May, 2006, at 12 o'clock noon.

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelyteorian laboratorio

Distribution:
Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O.Box 5400
FI-02015 TKK, FINLAND
Tel. +358 9 451 1
Fax. +358 9 451 3369
E-mail: lab@tcs.tkk.fi
URL: http://www.tcs.tkk.fi/

ISBN 951-22-8186-4
ISSN 1457-7615

Picaset Oy
Helsinki 2006

Abstract: Content delivery over the Internet to a large number of mobile users offers interesting business opportunities for content providers, intermediaries, and access network operators. A user could receive, for example, music or a digital newspaper directly to a mobile device over wireless networks.

Currently, content delivery over the Internet is held back by a number of reasons. Existing network technologies, such as GPRS, have a very limited capacity to transfer large files, such as those required for good-quality pictures in a newspaper. Another problem is security. Content received over the Internet is very vulnerable to being forged. A user who cannot be certain about the source and consistency of the received stock quotes is unlikely to pay for the information. Furthermore, content providers are unwilling to distribute their valuable information over the Internet due to their fear of copyright infringements. Traditionally, content has been considered consumed as soon as it has been downloaded. Content providers have been keen on preventing their content from being transferred over peer-to-peer networks because they consider the delivery itself to be a copyright infringement..

In this dissertation, content delivery is separated from content consumption by encrypting the content before delivery. When the users wishes to consume the content, a license which includes the decryption key is provided. The architecture allows content to be delivered to users' devices even before the user commits to consume the content. The user can choose to receive content whenever downloading it is the most convenient and affordable. Thus, the content providers are able to maintain control over the use of their information even after the data has been transferred to the users' terminals. In addition, content received by users can be strongly source authenticated.

The architecture allows secure, efficient and reliable delivery of content to a large group of receivers. The architecture does not commit itself to any specific delivery technique, and the content can be delivered using any delivery technique including multicast, broadcast, unicast, and peer-to-peer.

This dissertation focuses mostly on multicast as the delivery technique. The efficiency of the multicast delivery over unreliable heterogenous wireless access networks is thoroughly analyzed. Mobile terminals can seamlessly switch between access points and access technologies while continuing to receive data reliably from the network. The multicast delivery uses adaptive error correction and retransmissions to deliver the content as efficiently as possible to a very large number of receivers. The simulations show, that the vast majority of receivers are able to receive the content reliably with a small delay even when the radio network suffers from high packet loss probability.

Although the architecture is designed to deliver content to mobile terminals, it is also suitable for delivering content to terminals with fixed Internet connectivity.

Keywords: multicast, mobility management, wireless networks, secured content delivery, digital content delivery, digital content integrity

TIIVISTELMÄ: Digitaalisen sisällön siirtäminen liikkuville käyttäjille Internetin yli tarjoaa uusia liiketoimintamahdollisuuksia niin sisällöntuottajille, välittäjille kuin verkko-operaattoreille. Teknikkaa voidaan käyttää esimerkiksi musiikin tai sähköisten lehtien välittämiseen käyttäjille langattoman verkon kautta.

Sisällön välittämistä Internetin kautta hankaloittaa yhä usea seikka. Nykyisin laajassa käytössä olevat verkkotekniikat, kuten GPRS, ovat liian hitaita siirtämään hyvin suuria tiedostoja suurelle määrällä vastaanottajia. Lisäksi väärennetyn tiedon välittäminen Internetin kautta on erittäin helppoa. Sisältö, jonka aitoudesta ja alkuperästä ei ole varmuutta, on usein arvotonta käyttäjälle. Sisällöntuottajat puolestaan ovat haluttomia käyttämään sisältönsä levittämiseen Internetiä mikäli digitaalisesti levitettävän sisällön kopioiminen ja oikeudeton kuluttaminen on liian helppoa. Perinteisesti sisältö ajatellaankin kulutetuksi jo sillä hetkellä, kun se on siirretty käyttäjän laitteeseen. Sen vuoksi sisällön tuottajat ovatkin käyttäneet paljon resursejaan estääkseen sisältönsä välittämisen vertaisverkoissa, koska jo pelkkää sisällön siirtämistä pidetään tekijänoikeusrikkomuksena.

Tässä työssä erotetaan sisällön siirtäminen sisällön kuluttamisesta suojaamalla sisältö salauksella ennen sen siirtämistä käyttäjille ja sallimalla vapaa salatun sisällön jakelu. Arkkitehtuuri mahdollistaa sisällön siirtämisen käyttäjien laitteille silloin kun sisällön siirtäminen on edullisinta ja tehokkainta. Vasta käyttäjän halutessa kuluttaa aiemmin lataamaansa sisältöä, tarkistetaan oikeis sisällön käyttöön. Arkkitehtuuri mahdollistaa myös ladatun sisällön alkuperän ja eheyden vahvan tarkistamisen.

Arkkitehtuuri mahdollistaa turvallisen, tehokkaan ja luotettavan sisällön siirtämisen suurelle määrälle vastaanottajia. Arkkitehtuuri ei pakota sisällön jakelua käyttämään mitään tiettyä siirtomenetelmää vaan sisältö voidaan siirtää käyttäen esimerkiksi ryhmälähetystä (multicast), joukkolähetystä (broadcast), täsmälähetystä (unicast) tai vertaisverkkoja (peer-to-peer). Tässä työssä on keskitytty analysoimaan ryhmälähetyksen soveltuvuutta tiedon siirtomenetelmänä.

Ryhmälähetysmenetelmän tehokkuutta on analysoitu siirrettäessä sisältöä heterogeenisen langattoman liityntäverkon yli. Liikkuvat päätelaitteet voivat siirtyä saumattomasti liityntäverkosta toiseen samalla kun ne vastaanottavat sisältöä. Ryhmälähetys hyödyntää adaptiivista virheenkorjausta ja uudelleenlähetyksiä siirtääkseen sisällön mahdollisimman tehokkaasti suurelle joukolle vastaanottajia. Simulaatiot osoittavat, että erittäin suuri osa vastaanottajista saa sisällön luotettavasti ja pienellä viiveellä vaikka liityntäverkossa pakettien virhetodennäköisyys olisi suuri.

Arkkitehtuuri on suunniteltu siirtämään sisältöä liikkuville laitteille, mutta sitä voidaan käyttää yhtä hyvin myös kiinteään verkkoon liitettyjen laitteiden kanssa.

AVAINSANAT: ryhmälähetys, liikkuvuuden hallinta, sisällön siirtäminen, langaton tiedonsiirto, digitaalisen sisällön eheys, digitaalinen sisällön jakelu

CONTENTS

List of Figures

List of Tables

LIST OF ABBREVIATIONS

ACK	Acknowledgment
ADSL	Asynchronous Digital Subscriber Line
ALC	Asynchronous Layered Coding
API	Application Programming Interface
CBC	Cipher Block Chaining
CD	Compact Disc
CSS	Contents Scrambling System
DDoS	Distributed Denial-of-Service
DoS	Denial-of-Service
DRM	Digital Rights Management
DVB	Digital Video Broadcasting
DVB-C	Digital Video Broadcasting - Cable
DVB-H	Digital Video Broadcasting - Handheld
DVB-S	Digital Video Broadcasting - Satellite
DVB-T	Digital Video Broadcasting - Terrestrial
DVD	Digital Versatile Disk
FEC	Forward Error Correction
GPRS	General Packet Radio Service
HA	Home Agent
HSCSD	High Speed Circuit Switched Data
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IPsec	Internet Protocol Security
IPv6	Internet Protocol Version 6
iTMS	iTunes Music Store
IV	Initialization Vector
MAC	Message Authentication Code
MD5	Message Digest 5
MLD	Multicast Listener Discovery
MP3	Moving Picture Experts Group Layer-3 Audio
MPEG	Moving Picture Experts Group
NACK	Negative Acknowledgment
NORM	NACK-Oriented Reliable Multicast
P2P	Peer-to-Peer
PDA	Personal Digital Assistant
PDF	Portable Document Format
PGMCC	Pragmatic General Multicast Congestion Control
QoS	Quality of Service
RFC	Request For Comments
RSA	Rivest Shamir Adleman
SF	Svenska Filmindustri
SHA1	Secure Hash Algorithm 1
SSM	Source Specific Multicast
TCP	Transmission Control Protocol
TESLA	Timed Efficient Stream Loss-Tolerant Authentication

TFMCC	TCP-Friendly Multicast Congestion Control
TTP	Trusted Third Party
UDP	User Datagram Protocol
VoD	Video on Demand
WLAN	Wireless Local Area Network

ACKNOWLEDGMENTS

This dissertation is the result of research at the Laboratory for Theoretical Computer Science from 2002 to 2005. Most of the work was done in the Brocom project which was funded by Tekes from 2002 to 2004.

I would like to thank the pre-examiners, professor Jarmo Harju (Tampere University of Technolgy) and docent Göran Schultz (University of Turku). Their comments and observations were very important to me while I was finishing this manuscript.

I would especially like to thank my supervisor professor Hannu H. Kari. He always had time to help me with my work during the last four years.

1 INTRODUCTION

The wireless technologies needed for delivering video streams or high quality audio files to wireless mobile devices already exist. It is possible to use UMTS or GPRS networks to download music or video clips to a wireless mobile device while on the road. However, this is currently neither practical nor affordable.

A typical four-minute high quality audio file requires approximately 15 minutes to be delivered to the wireless device over a GPRS network. A video clip of equal length will take much longer even if the audio and picture quality are not very good. A look at the price listings of the major mobile phone operators in Finland reveals that the transfer of a four-minute music file costs at around one euro. Depending on the operator and the type of contract, the price can be several times more. In many cases, the price will be close to five euros, and this price only covers the cost of transferring the file from the network to the mobile device.

Downloading a legal copy of a song from the Apple iTunes Music Store costs 99 cents. When this price is added on top of the transfer cost, the total cost of downloading a song starts at two euros. Even in the most affordable case, the cost of transferring the song to the mobile device constitutes 50 percent of the total cost. While the song may be worth its price to the consumer, the cost of transferring it to the mobile device is hardly reasonable.

An alternative solution to using the traditional telecommunications networks is to use existing wireless Internet technologies such as WLANs to transfer the data. This technology, too, already exists, and it can be used to transfer large amounts of data rapidly and affordably. However, this technology too has its downsides. While GPRS networks currently cover almost entirely the inhabited area of Finland, WLAN provides reasonable coverage only in densely populated areas. Even in cities, the coverage is less than perfect. Another problem in using the Internet for content delivery is security. The user must be able to verify the authenticity of the content received. On the other hand, the content provider needs to be adequately protected against illegal use of its content.

In this dissertation, I have designed and analyzed an architecture that can be used for distributing large files or streams to wireless mobile terminals. The goal is not to design a system which would completely replace the existing telecommunications network. Instead, the system is designed utilize the best properties of both the existing telecommunication networks and the new wireless network technologies such as WLAN.

1.1 Original contributions

I claim the following contributions. The location of each contribution is given in the parenthesis after the description.

1. An architecture which allows mobile devices to receive content reliably over an unreliable wireless network utilizing multicast data delivery. (Chapter 6)

2. Security enhancements to the architecture which provide content in-

tegrity and content ownership protection. (Chapters 6 and 8)

3. Reliability analysis of the architecture. (Chapter 7.4)
4. Performance analysis of the architecture. (Chapters 9 and 10)
5. A proof of concept Linux implementation of the architecture and the protocol using the multicast delivery technique between the wireless access point and the group of receiver nodes. The implementation is able to interoperate with existing applications, such as the Mozilla web browser and the MPlayer video player. (Chapter 7.2)
6. Standardization improvements. (Chapter 11)

1.2 Outline of dissertation

The rest of this dissertation is organized as follows. Chapter 2 defines the research problem being solved, while Chapter 3 lists the criteria used in evaluating the solution. Chapter 4 defines a generic model for an architecture that can be used to deliver content over the Internet, and Chapter 5 presents and discusses previous work in the research area. Chapter 6 presents the architecture, which is then analyzed in Chapter 7. The security of the architecture is analyzed in Chapter 8. Chapter 9 presents the analysis of the efficiency of the implementation of the architecture using multicast delivery. Chapter 10 presents the analysis of the additional energy consumption of the architecture. Chapter 11 presents modifications to existing protocols and networks that are needed to use the architecture. Finally, Chapter 12 concludes the dissertation.

2 RESEARCH PROBLEM

2.1 Research problem

In this work, *content* is defined as any valuable file or stream of bits that needs to be sent over a network. One possible example of content is a live television stream. A *content provider* creates content for distribution in the network.

Most devices that are used for receiving content from the Internet are currently using wireline technologies, such as ADSL. However, wireless mobile devices are becoming more popular, and delivering the same content to mobile devices is becoming more and more important. A *mobile terminal* is a device which is used for receiving content from the network. A mobile terminal may be physically moved from one place to another, and it may be a device that is carried by a user. Examples of mobile terminals include a portable computer, a Personal Digital Assistant (PDA), a mobile phone, and the traffic information system in a car.

A mobile terminal uses one or more *access networks* to communicate with hosts in a fixed network. An access network can use either wireless or wireline technology, and it is used for transferring data between the mobile terminal and the fixed network. A *heterogenous wireless access network* is a set of access networks which consists of several different wireless access technologies. A mobile terminal may use one or more different access technologies to connect to the Internet. These access technologies may have widely differing properties. On one hand, the throughput of GPRS is in the order of tens of kilobits per second while the network coverage is virtually complete over entire countries. On the other hand, the throughput of WLAN technologies are in the order of tens of megabits per second, but they have very limited network coverage, only in hot spots. A mobile terminal which is utilizing a heterogenous access network is capable of frequently switching between access technologies. This allows the mobile terminal to optimize its operation between several parameters, such as transmission cost, capacity and access availability. The mobile data networks, such as GPRS and UMTS networks are seen merely as possible access technologies that can be used to connect to the fixed network, such as the Internet.

Network devices

Any device which uses a computer network to communicate with other devices in the network is called a *node*. A node in the network which is used by the content provider to send content into the network is called a *sender*. A sender uses the network to deliver content to a group of *receivers*, which can be either mobile terminals or any other nodes in the network. A *user* is any entity in the network which uses a mobile terminal to receive content. A user not only receives the content but also uses the information delivered in it. The user can be, for example, a human being or a computer program which uses the information in the content as its input.

An *intermediary* in the network is a node which participates in the delivery of the content to receivers, but is not itself interested in receiving the information carried by the content. Examples of an intermediary include routers in a packet switched network as well as any possible cache servers which may temporarily store the content before it is delivered to receivers.

Any node in a network which belongs to one of the categories of sender, mobile terminal or intermediary, is called an *insider*. A node that is not an insider, is called an *outsider*.

Security

The security of the system can be divided into three categories. The first category is *integrity*. The mobile terminal needs to be able to verify that the content being received has actually been created by the content provider instead of an impostor. The mobile terminal also needs the ability to verify that all the information it receives is being received unmodified by a third party. The timeliness of the received data must also be verifiable. These properties are needed to protect both the user and the sender of the content. The user needs to be sure that the content being received has been created by an authorized sender, especially if the content is received in pieces from several places in the network. The sender, in turn, needs the confidence that an attacker cannot use the content delivery system for delivering incorrect or untimely content in its name.

The second category of security in the system is *availability*. That is, the system must be able to operate and withstand attacks by outsiders as well as insiders.

The third category of security in the system is *confidentiality*. The content that is distributed needs to be available only to those users who have permission from the content provider to use it. The permission to use the data may be received, for example, after payment.

Any node which attempts to disrupt the operation of the network is called a *hostile node*. Any node in the network can be hostile, including the sender nodes. An architecture which fulfills the requirements of integrity, availability, and confidentiality is defined as *secured*. As a whole, the term secured describes the ability of the system to resist the attacks of hostile nodes, and the ability of the system to continue operation while being attacked.

Reliability

In a *reliable* system, content is delivered in its entirety, without any loss or modification to the data between the sender and the receiver. That is, the entire file or stream of data that was sent by the sender can be exactly reconstructed by the receiver. Reliability is a property which cannot be completely guaranteed in a network, such as the Internet, where nodes can be disconnected from one another for any arbitrary amount of time. However, a reliable system is designed in a way which attempts to maximize the probability of the data being delivered in a reasonable amount of time. In the Internet, reliability of data transmissions have traditionally been ensured by TCP, which uses retransmissions to resend lost data. However, in a network where mobile terminals may lose connectivity for long periods of time, retransmissions, as used by TCP, do not work.

If the number of receivers who wish to receive the same content is large, the content can be delivered either using unicast or multicast technology. In the traditional multicast architecture used in the Internet, each member of the multicast group is considered equal and has equal rights and responsibilities. Each node in the group is allowed to send packets into the group, and the packets are delivered to every member of the group. *One-to-many* multicast is a special case of the traditional multicasting architecture. In one-to-many multicast, only one of the nodes in the multicast group is allowed to send packets to the group where the other nodes are listeners.

Efficiency
An *efficient* architecture delivers content to the receivers using as little network resources as possible. Several, potentially conflicting, criteria for evaluating the efficiency of a system are presented below.

The first category of efficiency is *wireless network efficiency*. The architecture needs to utilize as little resources as possible to transfer the content to the group of receivers. The efficiency at the wireless network is more important than the efficiency at the wired network.

The second category of efficiency is *delay*. The architecture needs to be able to deliver the content from the source to the group of receivers with a sufficiently small delay. The amount of delay that is acceptable varies depending on the content being transferred. Requiring a small delay may require that a larger amount of network capacity is needed for transferring the content. A tradeoff between the delay and the wireless network efficiency can be necessary.

The third category of efficiency is *energy overhead*. The architecture needs to be able to run on battery powered devices. The architecture must not require a significant amount of extra energy compared with transferring the same content over, for example, an unauthenticated TCP connection.

2.2 Problem statement

The main objective of this dissertation is to design an architecture for efficient, secured and reliable distribution of digital content to mobile terminals that are using heterogenous wireless access networks. The scope of this study is limited to one-to-many delivery of data in a multicast capable IPv6 network.

3 CRITERIA

The created architecture will be judged in Sections 7, 8, 9, 10, and 11 based on the following criteria.

Criterion 1 *Technology independence:* The designed architecture shall be able to utilize any existing and future wireless access technologies that can be used to transmit IPv6 packets, including unidirectional wireless links. The architecture shall be able to utilize a heterogenous wireless access network, where all access technologies do not have full coverage.

Criterion 2 *Reliability:* The architecture must be able to operate in an environment where wireless connectivity is not always available and packet loss is high.

Criterion 3 *Security:* The system must be able to operate even when it is under attack by hostile nodes.

Criterion 4 *Efficiency:* The system must use efficiently the scarce resources of the network, especially on the wireless links.

Criterion 5 *Business feasibility:* The system must be implementable, and it must be able to cooperate with legacy software which do not specifically have support for multicast delivery. The system must also be able to separate the transfer and the consumption of content. The architecture should also discourage users from distributing copyrighted information without content owner's control over who are allowed to consume the content.

Criterion 6 *Mobility:* The designed architecture must be able to deliver data to mobile devices that may frequently change their point of attachment.

4 GENERIC MODEL

In this chapter, we discuss one-to-many distribution of content from a very high level point of view, without committing to any specific technology. We discuss who are the business players using the system, what kinds of content can be delivered, what network technologies can be used, and what kinds of business models the system allows.

4.1 Business players

In the most simplistic case, a system for delivering content via a one-to-many channel consists of a content provider and users. The content provider is interested in using the system to deliver its content to a potentially large number of receivers. The content provider assumes the content it is producing is valuable and the users of the content are willing to pay for it. [1] While the content provider wants the data to be easily accessible to as many users as possible, it also wants to protect its valuable data against unauthorized use. That is, the content provider needs to be able to protect itself against unauthorized use of its content.

If the content providers have data that the users find interesting, the users are willing to pay for it. This also assumes the user does have a terminal which can be used to consume the data. The price requested by the content provider also needs to be reasonable. Furthermore, obtaining a legal copy of the content should not be more difficult than obtaining an illegal one.

In practice, it is difficult for a user to purchase content directly from the content providers. Since the number of content providers is not limited in any way, each piece of content wanted by a user may be coming from a different provider. From the user's point of view, it would be much simpler to purchase all content from one *content distributor*. The distributor can purchase rights to distribute content from the content providers. The distributors then sell rights to users to use the content.

Figure 1 illustrates the relationship between the content providers, content distributors, and the users. Each content provider distributes its content through one or more distributors. The users should not need to get their content directly from the content providers but from the distributors. The users benefit from having to purchase data from only one source. Therefore, if the user needs, for example, some special smart card for using or paying for the data, a card from only one distributor is required.

4.2 Content types

Several types of digital content, differing in their type of structure can be identified. The type of content affects the way the data of a certain type needs to be handled by the architecture. The types are file, stream, and structured file.

---

[1] In some cases payment is not required. For example, the YLE radio and television broadcasts in Finland.

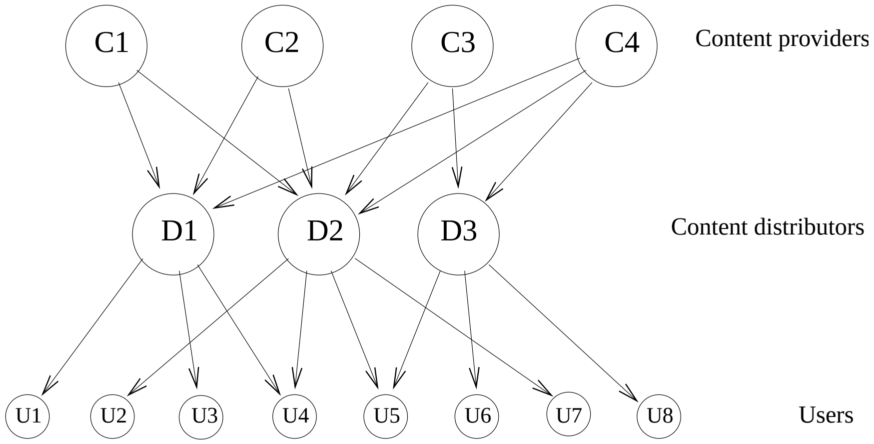

Figure 1: Relationship between users, distributors and content providers.

File

The file is the simplest structure for content. A file is a sequence of bits, and it has a beginning as well as an end. It almost always needs to be delivered reliably to the user to be valuable, and it cannot have bit errors. Typical examples of files include MP3 files, PDF documents, and web pages.

Stream

The stream, too, is a sequence of bits. However, unlike a file, a stream does not necessarily have a beginning, an end or a known length. The data in a stream is structured in a way which allows a terminal to start utilizing the content starting at almost any point in the stream. A stream can be either reliable or unreliable. In an unreliable stream, some data belonging to the stream can be lost, and the user of the stream must be able to tolerate the loss. On the other hand, a reliable stream guarantees that all data, after the point where reception of the stream was begun, is received. If a reliable stream is needed, the delivery architecture needs to use either retransmissions or forward error correction techniques to repair to content. However, repair mechanisms can be used even for delivering an unreliable stream to improve availability.

Examples of uses for an unreliable stream include a live television broadcast which is viewed in real time on the mobile terminal. If a small number of packets are lost from the stream, the software in the mobile terminal may be able to hide the loss of data sufficiently well that a user never sees that packet loss has occurred. On the other hand, it can be useful to use reliable stream delivery when the data is not viewed immediately, but it is stored to be viewed later. In this case it can be preferable that a perfect copy of the delivered stream is received. Then, if retransmissions are required to repair the stream, no delay is perceived by the user even if the lost packets are retransmitted much later than their position in the stream would imply.

Structured file

A structured file is a file which may consist of any number of subfiles or streams. A subfile can also have subfiles of its own to any depth. Together the subfiles comprise a structure which can be received from the network in parts, when data is needed. Each subfile can be updated independently from other subfiles.

The structured file allows a terminal to download only the necessary parts of a large structure and allows it to be easily and regularly updated. For example, an electronic newspaper can be delivered as a structured file. If capacity is very limited when the paper is delivered, only the headlines and most important articles need to be received. Large pictures and less important articles can be delivered only when the user specifically requests them. Also, articles in the newspaper can be updated in real time and they can be automatically delivered without reloading the entire newspaper.

4.3 Mobile terminal

The mobile terminal is the device which users use to consume content created by the content providers. The basic network connectivity requirements for mobile terminals are very small. The only connectivity requirement for a mobile terminal is the ability to receive data over one wireless network interface, which may even be unidirectional. The abilities of such a device is, however, very limited. A terminal with only unidirectional connectivity cannot be used to request data from the network. The terminal can only be used to receive data that is already being sent in an area covered by the wireless link technology used by the device. Such a device is very similar to a traditional radio or television receiver. The user can choose which transmitted data is received but the user cannot request data to be transmitted.

A more advanced mobile terminal also has the ability to send data over a wireless channel. The channel can be used for requesting data, which might not otherwise be sent, from the network. The mobile node can request retransmission of data that was lost when it was transmitted earlier. It can also request content which would otherwise not be transmitted in the area where the mobile node is currently located.

Another advanced feature in a mobile terminal is multiple radio technologies. A mobile terminal which can utilize more than one radio technology has the additional option of selecting the best possible service for different situations.[2] Switching between technologies can help the mobile terminal optimize between variables such as cost and capacity.

4.4 Delivery

Television is one of the most widely used one-to-many content distribution channels in existence. The delivery technique most commonly used in television is broadcast over radio channels which are specifically reserved for each television station. However, one-to-many distribution of data does not necessarily limit the distribution technology to a broadcast. For example, it is quite

[2]One radio technology, such as GPRS, can have a cell radius of kilometers, but only a limited capacity on the order of 40kbps. Another technology, such as WLAN, can have a cell radius of only tens of meters and a capacity of 54Mbps. It is probable that network coverage for the technology with the smaller radius is available only in hot spots, while the service with the larger cell radius is available practically everywhere.

possible to use a TCP connection to send live television over the Internet. While TCP is usually not the best alternative, it is certainly one possibility and demonstrates that other technologies besides broadcast are possible. The other possible high-level alternatives to wireless distribution technologies include at least broadcast, unicast, peer-to-peer (P2P), and multicast delivery.

Broadcast is the oldest and the most commonly used of the techniques. In broadcast, all data that is delivered through the wireless broadcast network is sent over the radio channel even when no users are interested in the content. On the other hand, if the number of interested users is high, all users can receive the information from the one transmission. Optimally, no data is sent more than once in an area. Broadcast can be efficient if the number of receivers in an area is high. However, when broadcast is used to send less popular content, the result is unnecessary use of radio spectrum. Furthermore, if reliable delivery is needed over an unreliable medium and a return channel is not available, the sender has no guarantees that the content has been completely received when it stops the transmission.

Unicast is a delivery technique where each terminal receives content over a connection dedicated to delivering data only to itself. A server can simultaneously send content to any number of terminals, but each additional terminal requires an additional connection and transmission resources at the server. The content can be sent to the terminal, for example, using TCP or UDP over the Internet. If very popular content needs to be delivered simultaneously to a large number of terminals in one area, unicast delivery will incur unnecessary use of network capacity since each terminal is allocated a separate connection. Additionally, the network capacity required at the central server increases rapidly as the number of receivers increases. Unicast is therefore not a suitable technique for delivering content consisting of very large files or high capacity streams.

P2P is a generalization of the traditional unicast delivery. In P2P, data is usually delivered through unicast connections. However, no centralized or official servers are needed, but any node in the network can deliver the data to another node in the network. The problem in using P2P is the additional load to the terminals who are also distributing content to other terminals. Even worse, the receiver cannot usually verify the original source of the information received over a P2P network. The content received over a P2P network may have been modified at any node in the distribution chain. The problem of tampered data is not specific to only P2P networks. However, the store and forward method of delivery makes the attack especially simple in P2P networks.

Multicast delivery strikes a promising balance between unicast and broadcast techniques. On one hand, data is only sent in areas when there are nodes which are interested in receiving the data. This property cuts down on unnecessary capacity use at the wireless interface. On the other hand, multicast also allows every receiver in an area to receive the same data from one transmission. Optimally, multicast combines the best properties from both unicast and broadcast.

It is noteworthy that the delivery network for transmitting the data does not need to have the same hierarchy as the network used for selling and purchasing authorization for using the data. Data can be delivered to a terminal using any of the technologies presented here. Even network technologies yet to be invented can be used as long as the data is eventually delivered to the terminal.

4.5 Delivery vs. consumption

When a traditional product such as a car or a computer is built, the cost of creating each individual copy of the item is significant. Not only has the manufacturer spent a large amount of money in designing the car, but a large amount of resources are spent in the building of each copy.

In digital content, the situation is different. The cost of creating new content can be very high, but once the first copy has been created, additional copies can be made with little or no extra cost. That is, the incremental cost in making a copy for each customer is virtually zero. This is especially true when the content is delivered through a network, such as the Internet, as nothing physical is produced when a new copy is made.

When a physical product is stolen, the owner of the product loses the ability to use it. The victim must either purchase a new copy of the product or live without it. In the digital environment, the equivalent of theft is the act of making an illegal copy. The difference is, that the victim does not usually lose the original product, but a copy is made by the thief. Since the victim does not lose anything when the copy is made, the product can hardly be considered stolen. As a result, the user has very little interest in protecting the content from being copied. In fact, it is usually in the interest of both of them to cooperate, since each can make copies of interesting content the other one has. In the end, both of them benefit and neither one of them sees the other one as a criminal. This is what makes peer-to-peer file sharing popular among users and hated by content providers.

In fact, the real problem for the content provider is not illegal copying but unauthorized use. No loss is incurred on the content provider when additional copies are made. As long as the copy can be only used after paying for a license, copying can be seen as an effective way of distributing content. The real problem for the content provider is therefore in preventing users from using content without proper permission from the content provider. [3]

The content provider needs the ability to control who is able to use its content. The traditional way to achieve this has been copy protection. However, if content cannot be consumed in a device until authorization has been received from the content provider, copying is not a problem, even for the content provider.

4.6 Business model

An architecture designed for one-to-many multicast delivery of data enables direct business possibilities for access network operators, content providers and terminal manufacturers.

[3] For example, the content provider may be willing to give authorization for the user to burn a CD which will work in the user's car stereo. However, at the same time, the content provider might like to prevent the user from burning a CD that will work in the CD player of another user.

Network access

Access network operators provide the terminal with connectivity to the fixed network. Current mobile phone based technologies such as HSCSD and GPRS can be used for delivering content without modifications. In these technologies, billing has been implemented and they are available practically everywhere within the developed world. These system are, however, far from practical and affordable. Billing in these systems is usually based on the amount of transferred data. If large files need to be transferred, the cost of transferring files can be very high.

On one hand, technologies such as WLAN allow much higher throughput for user data than the technologies based on mobile phone networks. On the other hand, billing for WLAN access is still difficult. Such systems do exist, but in many cases they use complicated methods of payment and are far from being affordable. [4] In many cases, payment is not based on the amount of data transferred, but on buying access for a specified amount of time. Such payment methods are unsuitable for a system where a terminal may spend only a very short amount of time at one individual access point.

A system that bases billing on the amount of transferred data and requires little or no interaction from the user is needed. To be useful, the system should also allow several access point operators to bill the customer through one banking transaction.

Content providers and distributors

Another opportunity for business using the system is for the content providers. The content providers use the system for distributing their content over a network. From the point of view of the content provider, their business is to sell rights to use content they have created. The rights to use the content is not usually sold directly to the users, but to content distributors. The content distributors then sell the rights to the consumers. The distributors are needed to provide their users with a simple way of purchasing their content from only one source, instead of a large number of content providers.

The data can be distributed to the users using any existing data delivery method, including multicast, unicast, and peer-to-peer networks. However, as the content provider will usually want the information delivered as rapidly as possible to the users, an efficient delivery architecture is needed.

Using a network, such as the Internet, for distribution of content can lower the threshold in becoming a content provider. For example, no expensive investments are needed for starting a new television channel which is delivered only over the Internet. Furthermore, the feasibility increases of creating content directed at very small niche markets. The users of content produced by a content provider do not need to be located in one country only. Since distribution of content over a network is practically free, the users can be spread all over the world. Even if the number of users in one country is small, the other users from around the world can make a niche product interesting for a sufficient number of users.

[4] An example of a system falling into this category is HomeRun [6] from TeliaSonera.

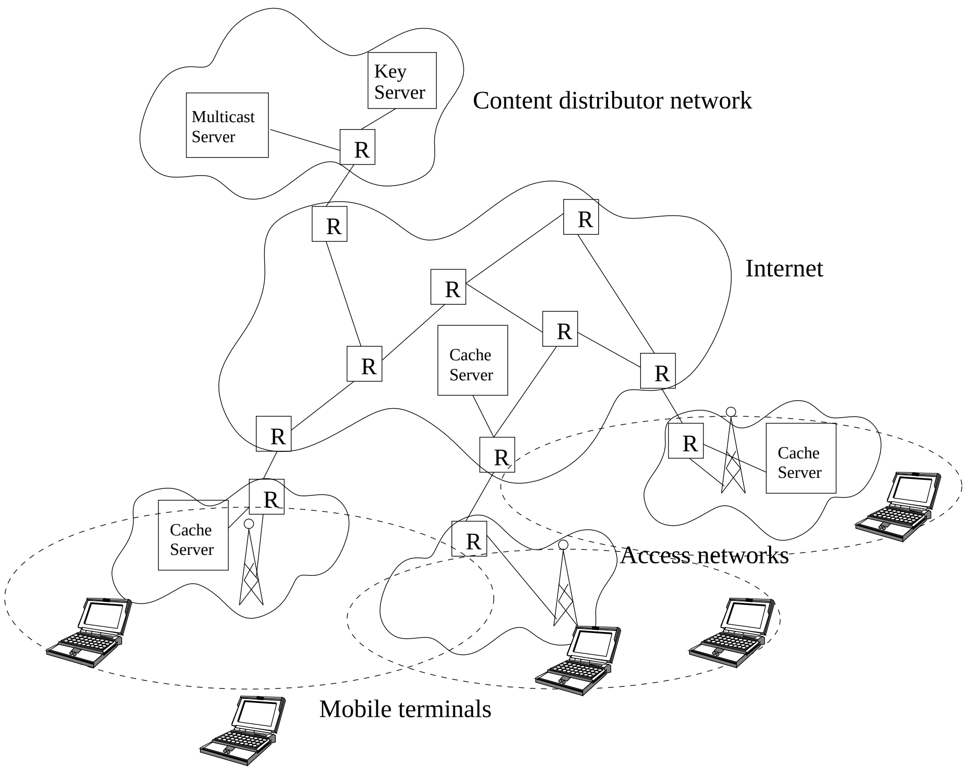

Figure 2: High level architecture.

Terminals
Terminals are yet another business opportunity created by this architecture. The terminal manufacturers can build devices for several different groups of users. The first type of terminal is the mobile terminal, such as a mobile phone, a PDA, and a laptop computer. Such devices communicate with the network over one or more wireless network interfaces and are typically carried around by their user. Another possible type of terminal has only wireline network interfaces. This kind of device can be used in homes, for example, to receive television over the Internet. The terminal with only wireline network interfaces is a simplified special case. An architecture which is able to transfer data over a wireless interface can also be effectively used over a wireline connection.

4.7 Generic architecture

Figure 2 illustrates the generic architecture developed in this chapter. The figure shows the relationships between the distributors, the Internet, the access networks, and the mobile terminals. The boxed R-letters in the figure are routers which connect the network together. The towers with dashed ellipses are wireless access points with the range of their radio technology shown. The cache servers are points of temporary storage where content can be stored to facilitate retransmissions to the terminals.

For simplicity, the figure shows only one content distributor, although the number of content distributors is not limited in any practical way.

5 BACKGROUND

This chapter presents background information relevant to the architecture which is developed in this dissertation.

5.1 Business

A number of companies in Finland are using Internet protocols for delivering video and audio content to users.

The datacommunication company Ålcom [1] offers its customers the ability to use Video on Demand (VoD) service over the Internet. The company uses the film archives of the Swedish movie distributor SF Anytime. Ålcom delivers the movies over its ADSL and ADSL2 network. The cost of watching a movie is roughly the same as if the movie was rented from a video store. The user is allowed to view the movie any number of times during the 24 hour period starting at the time when the movie was ordered.

Another Finnish company delivering television over the Internet is Maxisat [4]. The company is selling Internet access over ADSL. One of the extra services offered is the ability to receive television channels over the ADSL network. The data is delivered as an MPEG-2 stream, and it can be viewed either on a computer or on a television. Customers can also listen to radio stations over the network.

5.2 Wireless access technologies

Several standards have been developed for wireless access technologies. We describe the properties of some of the most relevant ones of the currently available wireless technologies as well as some technologies which are currently being standardized.

IEEE 802.11b

The most widely used WLAN standard is the IEEE 802.11b [8], which specifies the physical and link layers for a protocol in the unlicensed 2.4GHz band. The theoretical maximum throughput of this standard is 11Mbps, and the maximum transmission range using omnidirectional antennas is about 100m. In practice, the 100m range requires line-of-sight (LOS) between the communicating devices, and the 11Mbps throughput can be achieved only under favorable conditions.

IEEE 802.11a

IEEE 802.11a [9] is an evolution of the earlier IEEE 802.11b. It operates in the 5GHz area of the spectrum, with a maximum theoretical throughput of 54Mbps. Another advantage in IEEE 802.11a is the larger number of independent radio channels. However, the maximum range of 802.11a is smaller than the range of 802.11b, especially indoors.

IEEE 802.11g

The IEEE 802.11g [10] is another WLAN standard operating at the 2.4GHz frequency. The theoretical maximum throughput for the standard is

54Mbps. The 802.11g standard shares the operating frequency with the 802.11b standard and offers similar cell sizes. The 802.11g was specifically designed to be backwards compatible with IEEE 802.11b.

IEEE 802.16

The IEEE 802.16 standards, also known as WiMAX, define radio interfaces for point to multipoint access technologies, which are designed to provide wireless Internet access for metropolitan areas. The basic 802.16 standard was designed to provide wireless links between base stations and exterior antennas in buildings. The exterior antennas are then connected to the networking infrastructure within the building [23]. The standard was originally designed to operate in the frequencies between 10 and 66 GHz, with line-of-sight propagation. The standard users channels of 20MHz to 28MHz wide, with throughput up to 134Mbps. A typical cell radius for 802.16 is expected to be between 1 and 3 miles.

The standard was later amended to 802.16a. The amendments include the addition of the frequency band of 2-11GHz, and the ability to communicate in non-line-of-sight (NLOS) conditions. The throughput of 802.16a is up to 75Mbps using channels 20MHz wide. The typical cell radius is expected to be between 1 and 3 miles, with a maximum cell radius of 30 miles. The standard is designed for both fixed and portable devices. Currently, Suomi Communications Oy [5] is offering Internet access over 802.16a in the Helsinki region in Finland.

The IEEE 802.16e is another amendment to the 802.16 and 802.16a standards. It is another standard designed for frequencies below 6GHz with NLOS capabilities. It also adds mobility support to the standard. Throughput for 5MHz channels is up to 15Mbps.

IEEE 802.20

The IEEE 802.20 [11] is a standard for Mobile Broadband Wireless Access (MBWA). It specifies a radio interface which is optimized to deliver IPv4 and IPv6 packets between mobile nodes and a base station. The standard supports several classes of vehicular mobility up to 250km/h. The requirements for the protocol include a peak downlink throughput of more than 1Mbps, and a peak uplink throughput of more than 300kbps. Cell sizes are expected to be "appropriate for ubiquitous metropolitan area networks and capable of reusing existing infrastructure." The standard will have NLOS capability both outdoors and indoors, and it uses the frequencies below 3.5GHz. The specification will include handoff support to maintain connectivity when moving between cells, systems, and frequencies.

DVB-T and DVB-H

The Digital Video Broadcasting (DVB) standard [7] can also be used to transmit IP packets. The DVB-T standard is used for terrestrial transmissions, and can deliver data even to mobile devices. The technology is unidirectional, and can only be used for delivering data in the downlink direction. The maximum throughput in an 8MHz channel can be as high as 30Mbps, and the cell radius can be as high as 100km.

The DVB consortium has also designed a standard targeted specifically

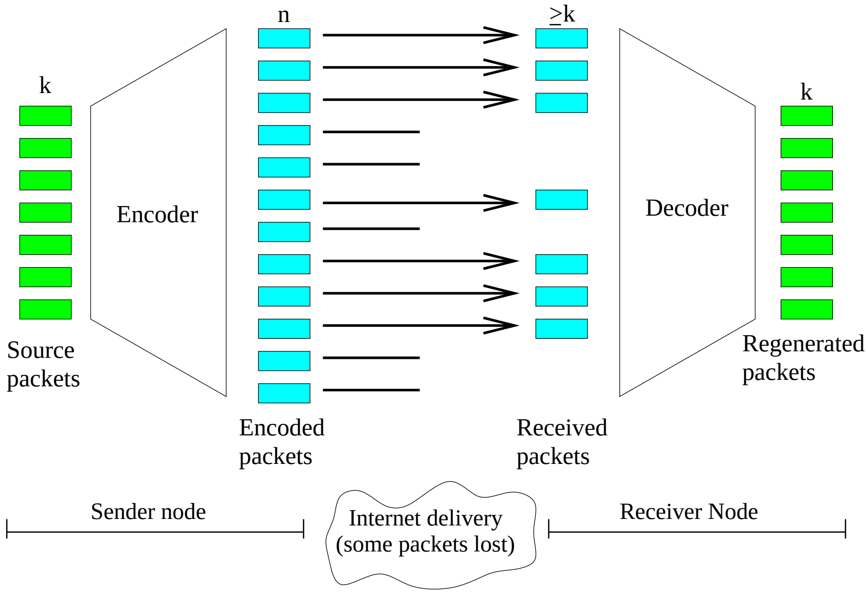

Figure 3: Example of an erasure code.

for handheld devices. The standard is be called DVB-H [12]. The standard is fully backward compatible with DVB-T, but adds support for handoffs between base stations. The standard is also be optimized to maximize battery life in handheld devices.

5.3 Erasure codes

An erasure code is a forward error correction method which can be used to recreate lost data packets using specially created error correction packets. An (n, k) erasure code, where $n \geq k$, encodes k blocks of source data into n blocks of encoded data. The source blocks and the encoded blocks are usually assumed to be of equal size. The key idea behind erasure codes is, that the source data can be completely reconstructed when any k blocks of encoded data are available to the receiver. That is, $n - k$ blocks may be lost without affecting the decoding process. Figure 3 illustrates the operation of erasure codes.

Erasure codes can be effectively used in reliable multicast delivery over packet switched networks. The sender generates n packets of data out of the original k packets that need to be delivered to the receivers. The generated packets are sent to the network and delivered to the group of receivers. Each of the receivers may receive a different subset of the encoded packets due to packet loss, but as long as at least k packets are received by each receiver, the receivers can reconstruct the original k packets. The erasure code described in [40] is used as the Forward Error Correction (FEC) algorithm in the architecture described in Section 6.

Further information and examples of erasure codes can found in [29] and [31].

5.4 Multicast

The model of multicasting used in the Internet was originally described by Deering and Cheriton in [19] and [21]. These papers describe an extension to the unicast delivery mechanism where each packet sent to an IP address is delivered to several hosts instead of just one. The hosts using this model of communication are defined in these papers as a host group.

Any host in the network may join the host group by informing a local multicast agent of its willingness to receive packets sent to a particular host group. A multicast capable host is assumed to provide applications with the following abstract API needed for multicast communication. The API is used for communicating with the host's multicast agent, and the functionality of some operations have been simplified for brevity.

- The **CreateGroup** command requests the creation of a new host group. If the operation is successful, a new host group is created and the address of the created group is returned as a result. The host who invoked the operation is now the only member of this host group
- The **JoinGroup** command is a request by the invoking host to become a member in the host group given as a parameter.
- The **LeaveGroup** command is the opposite of the JoinGroup command. It requests the invoking host to be removed from the host group.
- The **Send** command is used for sending a packet of data into a host group.
- The **Receive** command is used for receiving data from a host group.

The abstract API is front end to the IGMP protocol which is used by the hosts to communicate with its current multicast agent. The original IGMP protocol was defined in [20], whereas the most recent version of the protocol for IPv4 is defined in [16] and in [44] for IPv6.

The hosts in the host group are assumed to be equal, that is, any node in the host group is allowed to send packets into the group. From the point of view of the hosts in the host group, each packet sent to the address of the host group is delivered to every host in the host group. However, there are no guarantees that a packet will eventually reach every member of the group due to the best effort nature of the Internet. The routing infrastructure of the Internet hides the internal operation of multicast routing within the network

Since the original papers were published, terminology has somewhat changed. In modern terminology the *host group* is known as a *multicast group*. Also, the term *multicast agent*, used in the papers, has been replaced by the term *multicast router*. In the rest of this dissertation, the modern multicast terminology will be used.

5.5 Application layer multicast

Application-layer multicast is an alternative to network-layer multicast. Instead of implementing multicast in routers within the network, the functionality is assigned to the hosts in the multicast group. The benefit of this modification is that multicast functionality os no longer required from the routers in the network. The multicast packets are delivered through unicast tunnels between the members of the multicast group.

Several application-layer multicast protocols have been suggested. Examples of protocols and algorithms can be found in [14], [18], [43] and [38].

5.6 Multicast source authentication

Source authentication algorithms which are used in unicast communication cannot automatically be used for source authenticating multicast communication. Such algorithms are usually based on the assumption that only the two communicating parties have access to a shared secret. When one of the parties uses the shared secret to authenticate data, the other party uses the same shared secret to verify the authentication. If nodes A and B authenticate their communication using key k, A can verify that the data actually came from B by verifying a Message Authentication Code (MAC) in the data, using the same shared secret k. Now, if the communication system is expanded to include a third node C, which also has the key k, the system breaks down. Node A can no longer assume that data authenticated with the key k originates from node B, because it may as well have been created by node C. The problem becomes even worse as the number of nodes in the communication increases. This problem makes IPsec unsuitable for multicast authentication.

Another problem for multicast authentication is that each packet needs to be authenticatable immediately when it is received. That is, each packet needs to contain all information that is needed to authenticate it. The alternative of receiving several packets and authenticating all of them together, for example, through one public key signature over several packets cannot be used. A system relying on authenticating several packets in one operation is vulnerable to a simple denial of service attack. Even one forged packet among the ones being authenticated is enough to make the authentication fail. An attacker sending packets with forged signatures to a multicast group could effectively force receivers to drop even packets which have been sent by a legitimate sender.

A better method for source authentication of multicast streams, known as TESLA, is discussed in [35]. The system is based on loose time synchronization between the sender and the receiver. In this system, the sender authenticates packets using a keyed hash algorithm. The system uses late release of authentication keys as protection against forged messages. Since the data is received before the server reveals the key which was used for authenticating the data, a receiver cannot forge messages even if symmetric algorithms are used for authentication. However, this results in a system where the data cannot be even temporarily stored in the network, because data that is received after the authentication keys have been released, cannot be properly authenticated. Thus, the authentication mechanism is suitable for authenticating the data to a storage host which is guaranteed to receive the data as soon as it is transmitted into the network. However, a terminal which later receives the data from a storage server, cannot anymore source authenticate the data.

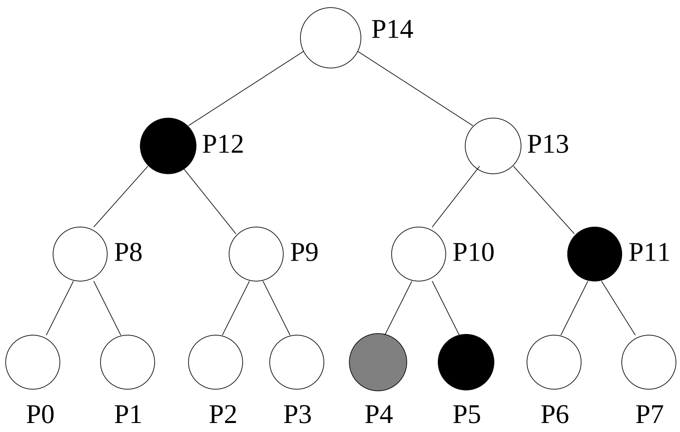

Figure 4: Hash tree.

The property which is really needed from the source authentication algorithm is nonrepudiation. Nonrepudiation is an additional property to a source authentication algorithm. As in any source authentication algorithm, the algorithm needs the ability to prove the sender of the data to the receiver. The additional property, which is also needed, is the ability to prove the identity of the original sender to any terminal that receives the data from the storage server. The simplest method of providing nonrepudiation is through signing the data using a digital signature algorithm.

The most simplistic method of signing multicast flows is to sign each packet separately from all other packets in the flow. However, such a system requires performing one public key signature verification every time a multicast packet is received. In this case, terminals with low computational capacity cannot be used for receiving multicast data.

In [47], a method for signing and verifying the sender of multiple packets using only one public key operation per a set of packets is introduced. The scheme is based on amortizing the signature on several packets using a tree based chaining technique. Even though multiple packets can be authenticated by only one signature, the system allows the receiver to authenticate the data immediately as it is received, even if other packets using the same signature are lost. The system also allows the receiver to verify the identity of the sender, even if the packet has been stored in a cache for any length of time.

Figure 4 illustrates the algorithm. The leaves correspond to hashes that are calculated over the packets to be signed. The value of each parent node is the hash of the concatenation of its children. For example, the value of node $P8$ is computed as $P8 = hash(P0|P1)$, where the vertical line denotes concatenation. When the tree has been generated, the sender signs the root of the tree. To enable the receiver to verify the signature, the sender must include into each packet, the hashes which are necessary to generate the path to the root. In the figure, the packet to be sent is $P4$. The sender must therefore include into the packet the hashes of nodes $P5$, $P11$, and $P12$. The sender also sends the signature of the root of the tree to the receiver.

When the client receives a new packet, it must regenerate the path to the root of the tree. In the case of Figure 4, the client first computes the hash over the received packet, which corresponds to leaf $P4$ of the tree. It then generates the parent node by hashing the newly computed $P4$ and the $P5$ that was received in the packet, that is, $P10 = hash(P4|P5)$. The receiver continues this process until it has created the hash of the root node. [5] The receiver then checks the signature to verify the validity of the hash of the root. If the verification succeeds, the packet is valid. A more detailed description of the algorithm can be found in [47].

5.7 Reliable multicast

Since IP networks provide only best-effort service, multicast delivered packets can be dropped, corrupted or rearranged as they are being transferred in the network. If the data still needs to be delivered reliably, a reliable multicast protocol is required. The reliability goals of reliable multicast protocols are similar to the reliability goals of TCP. If the path between the sender and the receiver has sufficiently low rate of packet loss, the data will eventually be transferred to the receiver. Several reliable multicast protocols have been created. We discuss only the two most significant protocols which are currently being standardized in the IETF.

ALC
The Asynchronous Layered Coding Protol Instantiation (ALC) [30] is a reliable multicast protocol published by the IETF. The protocol is designed to be massively scalable, and the expected maximum number of receivers for the transmitted data is in the order of millions. The protocol uses the FEC building block [32] for adding redundancy into the data, and no retransmissions are used in the protocol for recovering from packet loss.

Congestion control in the protocol is provided by splitting the transmission into multiple *channels*. When the Source Specific Multicast (SSM) model of multicast delivery is used, each ALC channel consists of one SSM multicast channel address. That is, each ALC channel is sent as a separate multicast channel. Each channel may transmit data at different rates. Receivers adjust their packet reception rate by joining a suitable number of channels. The number of joined channels is controlled by a congestion control building block, which is currently not specified in the RFC. The limitations for the congestion control building block include feedback-freeness. That is, the congestion control mechanism must not send any feedback to the multicast sender.

The combination of feedback-freeness and lack of repair retransmissions in ALC imply, that the multicast receivers do not need to send any data back to the multicast senders. This property gives ALC its ability to scale essentially without any limit to the number of receivers.

NORM
The NACK-Oriented Reliable Multicast Protocol (NORM) [13] is another reliable multicast protocol being standardized by the IETF. NORM uses both the FEC building block and retransmissions for ensuring reliable delivery. While ALC is designed to distribute objects, such as files, NORM has been

[5] For example, in Figure 4: $P14 = hash(P12|hash(hash(P4|P5)|P11))$

designed to deliver both objects as well as streams. In NORM, the number of receivers is expected to scale to the order of thousands. In ordinary data packets, FEC is not used by default. FEC is used only in repair packets which are sent in response to receivers reporting loss of packets.

When a member of the receivers detects the loss of one or more packets, a negative acknowledgment (NACK) packet is transmitted to the sender. The NACK packet is used to request retransmission of the lost data. In response to the NACK, the sender transmits a suitable number of FEC packets to the receivers to repair the data. The problem of NACK implosion is avoided through a backoff timer. When a receiver detects the loss of packets, it starts a timer with a suitable expiry time. The NACK packet is sent only once the timer expires. If suitable FEC packets to repair the data are received from the sender prior to the expiry of the timer, no NACK is sent. The reception of suitable FEC packets prior to timer expiry implies that some other receiver with a shorter timer has already requested retransmission, and the NACK implosion can be avoided by not transmitting unnecessary NACKs.

Another feature of NORM is the use of a congestion control mechanism where feedback is sent to the sender from the clients. There are two possible congestion control mechanism available to be used in NORM, the TCP-Friendly Multicast Congestion Control (TFMCC) [46] and Pragmatic General Multicast Congestion Control (PGMCC) [39]. Since the operation of these two algorithms are almost identical, only the operation of PGMCC is explained here.

The transmission rate in PGMCC is set to maintain TCP friendliness at lowest capacity link in the multicast tree between the sender and the receivers. That is, if even one of the receivers is located behind a low capacity link, all receivers will receive their data at the capacity dictated by the low capacity link.

The algorithm selects the receiver with the lowest capacity as the ACKER whose responsibility it is to send feedback (ACK packets) to the sender. Another type of feedback from receivers to senders in PGMCC is the NAK packet, which can be sent to the sender by any receiver. NAK packets are only sent in response to packet loss. When a sender receives a NAK packet, it may choose to select the sender of the NAK packet as the new ACKER. The ACKER is changed if the feedback indicates that the capacity between the NAK sender is significantly smaller than between the multicast sender and the current ACKER. Throughput between the sender and a receiver is estimated based on the round trip time (RTT) and the packet loss rate. Both are received in the NAK packets sent by receivers.

The security implications of relying on feedback from the group of receivers is well acknowledged in the specifications of both PGMCC and TFMCC. Both algorithms are vulnerable to an attack where any node in the network can send forged feedback data to the sender node. The simplest attack is to simulate a receiver whose network capacity suddenly drops down to a very low level. Such an attack can be used to bog down the transmission rate to all users in the group to essentially zero. It is practically impossible to determine whether or not a congestion report is really an attack on the system or not. Both PGMCC and TFMCC essentially assume that every receiver in the multicast group is benevolent enough not to send forged feedback to the sender. It is difficult to believe such a system can be used in a public network, such as the Internet, unless attacks on the system can be made more difficult.

Tradeoffs in reliable multicast

Feedback mechanisms in reliable multicast protocols are very simple to misuse. In both TFMCC and PGMCC, an attacker can easily limit the transmission rate of the entire multicast group to a halt by simulating a badly congested link. The problem with feedback mechanism is, that the multicast sender must rely on every one of the receivers to report correct values in their feedback messages. The specifications of both TFMCC and PGMCC suggest that feedback should be only accepted from the receivers. Even if doing this is feasible, it is not possible to trust the feedback, especially if the number of receivers is in the order of thousands. If a protocol using feedback is utilized to distribute, for example, TV-broadcasts over the Internet to customers, the receivers should not be trusted any more than any other user on the Internet. Unless the problem of forged feedback messages is solved in a satisfactory way, a feedback free reliable multicast protocol is needed when distributing data to a large group of users.

Another problem in using the existing protocols in receiving multicast to mobile terminals is the built-in limitations in existing protocols. The protocols used for joining and leaving multicast groups, IGMPv3 [16] and MLDv2 [44], do not support unidirectional last-hop link technologies. That is, according to the specification, join requests must always be sent over the same link which is used to receive the multicast data. This property excludes, for example, the usage of the unidirectional DVB-T technology as the last-hop link technology for receiving IP multicast data.

5.8 Real-time Transport Protocol

The Real-time Transport Protocol (RTP) [41] is a protocol for delivering real-time data over the internet. The protocol is designed for data where timely delivery is more important than reliability. Natural applications for the protocol are therefore audio and video delivery, optionally using multicast. The protocol is used over the UDP protocol and adds information about packet ordering and timeliness to UDP. The two most important fields in the RTP header are the following:

- Sequence number. The field enables the receiver to detect out-of-order delivery and lost packets.
- Timestamp. The field gives the receiver the ability to play the contents of the packet at an appropriate time, regardless of the changes in the delivery delay experienced by the packets.

RTP is accompanied by the Real-time Control Protocol (RCTP), which is defined in the same specification as RTP. RTCP provides RTP with out-of-band information about an RTP packet flow. Its primary function is to provide feedback of the RTP data distribution.

One of the fundamental assumptions in RTP is that all data needs to be delivered to the receivers with as little delay as possible. The protocol is build to detect packet loss and out-of-order delivery. However, the protocol gives no guarantees of reliable delivery. Packet loss can be detected by the receivers, but if a packet is lost it cannot be requested to be retransmitted.

5.9 Secure Real-time Transport Protocol

The Secure Real-time Transport Protocol (SRTP) [15] and the Secure Real-time Control Protocol (SRTCP) are extensions to the RTP and RTCP protocols. The extensions provide confidentiality, message authentication, and replay protection to the protocols.

The authentication in SRTP is provided by a shared key using a symmetric key approach. The symmetric key needs to be made available to every member of the group which needs to be able to authenticate the received packets. However, symmetric key authentication only ensures that the message has been created by a node which has acces to the shared key. Hence, any member of the group is able to create forged messages. A couple of alternatives are currently being developed by working groups in the IETF.

TESLA authentication

TESLA is offered as an authentication mechanism for SRTP in [36]. TESLA itself was discussed in Section 5.6. The algorithm is based on releasing the authentication keys after every receiver has already received the packets being authenticated.

The algorithm is lightweight but it has a significant problem. The packets lose their authentication if they are not received by their authenticator immediately after they were sent. This property makes TESLA useless in any scenario which allows data to be stored by one or more intermediaries before being delivered to the eventual receiver. Such a property makes, for example, P2P delivery and caching in the network impossible.

RSA authentication

In [45], RSA signatures are suggested as an authentication mechanism for SRTP. The solution is to sign each packet with an individual RSA signature. Since packets are signed using public key cryptography, the authentication can prevent forgeries by receiver group members. The packets also maintain their autentication even when they are stored by an intermediary, before being delivered to the receiver.

The downside of the mechanism is the high overhead. Since each packet is signed separately from every other packet, an RSA signature needs to be verified every time a packet is received. If keys of sufficient length are used in the authentication, the mechanism is likely to require computations that are too intensive for most mobile devices.

5.10 RTP caching

In [26], the authors suggest a caching architecture which stores RTP packets in caches. The paper does not focus specifically on caching data, but on caching the beginings of data streams to minimize startup delay. Furthermore, the paper does not mention security, multicast or mobility in any way.

5.11 Multicast and mobility

When a mobile terminal moves to a new network due to node mobility, any multicast streams the terminal was listening to will be terminated. The mobile terminal has two options which it can use for restoring the stream of data received through multicast.

The first alternative for restoring the multicast stream is to rejoin the multicast group from the new network. Immediately after the mobile terminal has arrived in the new network, it uses MLDv2 [44] to join the multicast group from its new address. This solution may be acceptable if the mobile terminal can tolerate potentially losing a large number of packets during the transition. However, if a stream of data with low packet loss is required, gaps in the coverage of a wireless network can make this technique infeasible.

It is possible to improve the tolerance of terminals against packet loss through reliable multicasting protocols, such as the ones presented in Section 5.7. However, these protocols are designed to protect against packet loss in the fixed Internet. As the loss of connectivity during transitions between networks can be arbitrarily long, such protocols cannot be relied upon for reliability.

The second alternative for receiving multicast data is to use the multicast data packet forwarding described in Mobile IPv6 [25]. In this alternative, the mobile terminal does not perform the MLDv2 functionality locally, but receives the multicast data through a tunnel from its Home Agent (HA). The mobile terminal uses the Home Agent to join multicast groups in its home network, and the data packets are tunneled to the mobile terminal from the home network.

However, tunneling from the Home Agent loses most of the benefits of using multicast delivery. Every mobile terminal requires a separate tunnel between itself and its Home Agent. Therefore, even when several mobile terminals are receiving the same data through multicast in the same area, the data is transmitted to every node through a separate unicast stream.

5.12 Digital watermarking

Watermarking is defined in [33] as the embedding of encoded information into digital data so that the information is imperceptible, easily read by authorized parties only, and difficult to remove by unauthorized parties without destroying the (value of the) original data. The paper lists several quality criteria for watermarking methods. For brevity, only the criteria most relevant to our architecture are listed here.

1. **Fidelity:** The changes entailed by marking should not affect the value of the content, and ideally the mark should be imperceptible. Specifically, experts should not be able to discriminate between the watermarked data and the original.

2. **Security:** Watermarks should survive deliberate attempts to remove them. Ideally, a watermark should remain readable up to the point

where the content becomes modified enough to by of low value.

3. **Accuracy of detection:** How accurately can the mark be read? What is the chance of of a false positive, a false negative, or a false reading?

In addition to an algorithm for writing watermarks, a watermarking system also needs an algorithm for reading watermarks from existing content. Examples of watermarking algorithms can be found in [22], [27], [48] and [28].

5.13 Digital rights management (DRM)

Several competing Digital Rights Management systems are currently being used by content providers. The goal of these systems is to make it as difficult as possible to make more copies of the content than the content provider allows.

The iTunes Music Store (iTMS)[3] is currently the most widely used paid online music service. iTMS allows users to purchase music at a reasonable price of $0.99 per song. Apple allows music downloaded from iTMS to be burned to CD an unlimited number of times. The user can also listen to the music on up to 5 computers or download it into any number of Apple iPods. The DRM scheme used by iTMS is called FairPlay. The encoded music files are always stored encrypted on a computer. However, it is not difficult to see that such a system is very simple to break. Since the iTunes application can be used on a general purpose computer with no built in decryption hardware, the decryption keys will be available in the memory of the computer when the song is being decrypted. The decryption key can then be retrieved from the memory and be used to decrypt the protection on the encoded file.

Soon a decryption program for music from iTMS was freely available on the Internet. In response, Apple modified the FairPlay system when it released version 4.5 of its iTunes software. Within 24 hours, the new version of the algorithm was also broken, and decryption software was again available to everybody. The current version of the decryption program is called Hymn. Documentation and download for the software is available at [2].

Another copy protection system is the Contents Scrambling System (CSS). It is the encryption system used to protect Digital Versatile Disks (DVD). The security of this system is based on storing a decryption key in the software or hardware of the player. Each manufacturer of DVD players is assigned a *player key*. The player key is a symmetric encryption key which is used to decrypt the *Title key*, which is used to decrypt the content on the disk. The title key is another symmetric encryption key, and it is stored on every disk encrypted using every player key. When a DVD is inserted into a player, the player uses its player key to decrypt the title key stored on the disk. The system is built in such a way to allow content providers to exclude players whose player key has been leaked. If a manufacturer's player key is leaked, content providers no longer encrypt the title key with the leaked key. That is, if a player key is leaked, new DVDs created after the key was revealed cannot be played by the players whose manufacturer's key was leaked.

Eventually one player key was extracted from the software implemented Xing DVD player. According to the DVD consortium's plans, players using the Xing player's player key would now be excluded from playing future DVDs. However, the CSS system was designed in a way which reveals the player keys of every manufacturer when one of the player keys was revealed. As a result, CSS became completely useless overnight. Details about the operation of CSS can be found, for example, in [42].

Both CSS and FairPlay could be broken because a decryption key was retrieved from the software implemented decryption algorithms. Aside from code obfuscation there is little that can be done to prevent a user from analyzing the operation of decryption software running in a standard computer. If prevention of such attacks is necessary, the decryption keys and the decryption algorithms need to be implemented in trusted, tamper proof hardware. Such hardware must never reveal the decryption key to the content and the decryption must also be implemented in the same tamper proof hardware.

Microsoft also has a DRM scheme called Microsoft Media DRM, and the system is included in the Windows Media Player 10. It is another DRM system based on a software implementation in the terminal. The system has been integrated deeper into the operating system than FairPlay. For example, the *Secure Audio Path* allows transmitting audio between applications and audio drivers in encrypted format to make the capture of unprotected audio more difficult.

Currently, no well known attacks that can remove Microsoft Media DRM from content seem to exist. However, as the technology is implemented in software, it will probably be broken as soon as enough interesting content is made available to encourage reverse engineering.

Problems in software DRM

DRM systems which are based on pure software implementations on the terminals are bound to fail in their current form. Even if the building blocks used to devise DRM schemes are strong, the DRM algorithms will continue to fail as long as the cryptographic primitives are used in unintended ways.

The way in which encryption is used in CSS, FairPlay, and Microsoft DRM are excellent examples cryptography gone wrong. Encryption is designed to protect the content of communication from being exposed to an outsider. That is, if nodes A and B are using encryption to protect their communication, node C who does not know the key, cannot eavesdrop on the content. However, in all of the above DRM schemes, encryption is used to protect the information from being revealed to the intended receiver. That is, nodes B and C are assumed to be the same node. Since encrypted content cannot be used until a decryption key is received, the key clearly needs to be given to node B. However, when the key is given to B, it is also given to node C, because they are the same node.

Since giving the key to the attacker is a necessary part of a software DRM scheme, the most basic assumptions of encryption are discarded. As a result, DRM designers rely on code obfuscation to protect the decryption key from being revealed to the users who could use the key for removing the DRM protection. Obfuscation is by no means a reliable security measure, and careful reverse engineering will eventually find a way to easily retrieve the key.

If encryption needs to be used to prevent unauthorized access to content, decryption needs to be performed inside hardware trusted by the content distributor. The hardware must never give out the key which is used to encrypt the data. Also, when the decryption key is delivered to the trusted hardware from the content distributor, it must be properly protected against being revealed. Finally, the device must be designed to be tamper proof to protect the keys from being extracted from the hardware.

6 THE ARCHITECTURE

This chapter introduces the architecture developed to satisfy the criteria described in Section 3. The system uses multicast to distribute the content into the network. The data is delivered to client devices using the existing multicast routing infrastructure of the Internet.

6.1 System overview

Content delivery in the architecture is divided into two parts: the fixed leg in a fairly reliable fixed network, and the less reliable leg in the access networks. The delivery in the fixed network distributes the data into the access networks as well as to a number of multicast caches. Some terminals may able to listen to the multicast transmission at the time when it is being sent. These terminals can continue to operate in the way of the standard multicast client. Other terminals which are unable to receive the multicast data directly, for example due to mobility into an area without network connectivity, can request the data to be resent. Retransmissions are handled primarily by multicast caches. When a terminal regains its connection to the network, it can request the lost packets to be retransmitted from the nearest multicast cache which is storing data of this multicast stream.

Another way used to improve reliability of multicast in our system is forward error correction (FEC). If necessary, redundancy can be added into the multicast stream. Then, the data remains usable to terminals even if a small percentage of the transmitted packets are lost. The FEC coding allows a terminal to regenerate the missing content.. The tolerance level against packet loss can be adjusted to optimize between the use of network capacity and the number of needed retransmissions.

Figure 5 illustrates our architecture. The use of the architecture is divided into five steps in the picture. [6]

1. The user selects content that the terminal should start to receive. The request is delivered to the multicast client in the mobile terminal using, for example, a web browser.

2. The multicast server sends the encoded content into the network. The content is received and stored by a number of multicast caches. If the mobile terminal is in a place where it is able to receive multicast streams while the content is being sent, the content is also delivered directly to the mobile terminal.

3. It is assumed that the mobile terminal has missed enough multicast packets that is cannot reconstruct the original content from the received packets. To reconstruct the content, the mobile terminal requests missing packets from one of the multicast caches which received the data in step 2.

4. The user decides to use the content. A decryption key needs to be obtained before the content can be used. The mobile terminal contacts

[6] The steps are not necessarily performed in the order given in the picture, and they can even happen simultaneously.

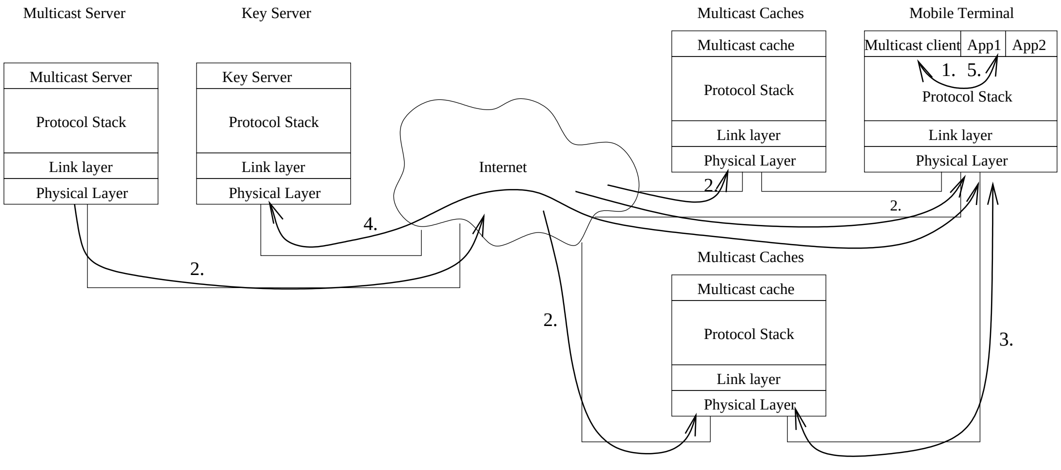

Figure 5: High-level view of the delivery architecture.

a key server and requests the decryption key for the content. The key server returns a decryption key to the mobile terminal. [7]

5. The client software in the mobile terminal reconstructs and decrypts the content. The content is given to an application through a local HTTP connection.

In our system, a multicast stream consists of a sequence of packets. Each packet in every multicast stream is given a unique *packet identity*. The packet identity can be used to request the packet to be resent if, for example, the original packet in the multicast stream was not received. The packet identity consists of the *multicast stream identity* and a sequence number. The multicast stream identity is specified by the source and destination address fields of the IP headers. Each packet is then uniquely identified within the multicast stream by a sequence number. A piece of content can then be simply identified by the combination of

1. A pair of source and destination addresses.
2. A range of sequence numbers.

The global identities are written as [<destination address>, <source address>, <start sequence number>, <end sequence number>]. For example, the global identity for a piece of music can be [FF1E::1234,3FFE::1,1234567,1236500], where the source and destination addresses are in IPv6 format. The same source and destination address pair can be reused as long as the sequence number ranges of the contents do not overlap.

6.2 Server overview

A high level overview of the stream generation algorithm is depicted in Figure 6. The roman numerals on the left hand side of the figure give the

[7] Before the key server returns a key, the user needs to prove its authorization to receive the key. This can be done, for example, through a certificate.

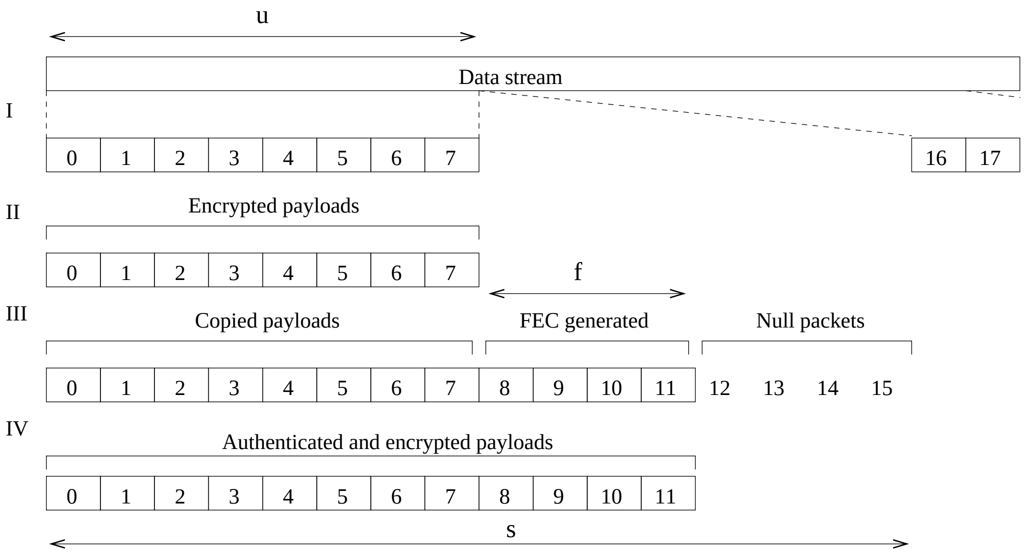

Figure 6: The steps in creating the packet stream.

number of each step in the algorithm. The details of the server operation are given in Section 6.4.

In step `I`, a block of data from the start of the unsent part of the content is divided into packets. [8] Note, that the next packets from the data stream will not necessarily be assigned the next sequence number, but a gap may be left into the sequence number space.

In step `II`, the packets are encrypted. Encryption is done using a block cipher in cipher block chaining (CBC) mode. The sequence number in each packet is used as the initialization vector (IV) for the cipher.

In step `III`, a number of error correction packets are created from the packets that were created in the previous step. Each error correction packet is also given a sequence number. The number of error correction packets can be anything from zero to the amount that fits between the last packet in the block and the start of the following data block. The only requirement is that the sequence numbers of the error correction packet do not overlap with the sequence numbers of the following block.

Finally, in step `IV`, the authentication information is added into the packets. The packets are then ready to be sent to the network after the IP header has been added.

6.3 Parameters

The multicast encoding and decoding algorithms require selecting values for a few variables. The variables control the amount of redundancy added into the packets and the number of packets from which the redundancy is computed.

An FEC block in a multicast data stream is a sequence of data packets and the FEC packets computed from those data packets. The width of an unen-

[8] In the figure, the block consists of 8 packets, but in practice the number of packets can be much higher.

coded FEC block, u, denotes the number of data packets used in computing the error correction data. The variable s denotes the number of sequence numbers that are allocated to be used for all packets within an FEC block. Finally, the variable f denotes the number of error correction packets generated for each FEC block. To make sure that all all error correction packets fit into the sequence number space allocated to them, $u + f \leq s$ must also hold. Also, the value for s must be a power of 2 to simplify computations.

The authentication algorithm also needs to have one additional parameter fixed. The height of the authentication tree, h, is chosen by the user, and it determines how many packets are source authenticated by one signature. From h, another parameter, the width w of the authentication tree is fixed. The parameter is defined as $w = 2^h$.

The parameters presented here are all fixed during the multicast session. These parameters will appear in pseudocode and text throughout the rest of this dissertation. In Figure 6, the parameters are set as $u = 8, s = 16, f = 4$.

6.4 Server algorithm

The multicast server generates a multicast stream from the content that it receives as input. The server reads in the data starting at the beginning of the stream. Steps from `I` to `IV` in Figure 6 are repeated by the server until no more input is available.

Before multicast data generation is started, a few variables need to be given starting values. At the beginning of the following pieces of pseudocode, q is set to be the sequence number of the first packet in the generated stream. The variable q also needs to be a multiple of s.

Packet stream generation

In step `I`, the multicast server generates u packets from the beginning of the unprocessed input content. Each of the packets are equal in size, and these packets will become the payload in the multicast stream.[9] The size of the packets needs to be small enough to fit the entire packet into the MTU of the network even after the lower layer protocol headers have been added to the packet.

Next, each packet is given a sequence number. The first packet taken from the remaining input content is given the sequence number q. Each following packet is given the sequence number one larger than the previous packet, until each u packets have been assigned a sequence number. The sequence number given to each packet is added into the headers in the packets.

```
int i;
for (i=0; i<u; i++) {
   packet[i].payload = dequeue(content_input, PAYLOAD_SIZE);
   packet[i].seqnum = q + i;
}
```

The pseudocode reads u packets from a queue which is holding content that is waiting to be encoded and sent into the network. A sequence number

[9] If packets are not of equal size, the smaller packets can be padded to the size of the largest packet in the FEC block before processing them.

is assigned to each packet.

Data encryption

In step `II`, the payload of the packets created in step `I` are encrypted. The encryption is done using a block cipher. The cipher is used in CBC mode, and the sequence number in the packet is used as the initialization vector for the cipher. The encryption covers only the payload part of the packet. The sequence number in the packet remains in plaintext. The payload is padded to a multiple of the size of the cipher before encryption, if needed.

```
int i;
for (i=0; i<u; i++) {
   pad_to_n(packet[i].payload, CIPHER_BLOCK_SIZE);
   cbc_encrypt(packet[i].payload, packet[i].seqnum, PAYLOAD_SIZE);
}
```

The pseudocode pads each one of the u packets into the size required by the cipher and encrypts the packets using the sequence number as the initialization vector. At the end of this step, the content of each packet will be as follows.

SeqNum, Encr(Payload | Padding)

That is, the packet consists of an encrypted payload and the sequence number.

FEC generation

In step `III`, f forward error correction packets are computed from the u packets generated in step `II`. The error correction data is computed over the payload of the original packets. That is, the sequence numbers in the packets are not included. Each one of the FEC packets are also given a sequence number, starting with the sequence number $q + u$.

```
int i;
for (i=0; i<f; i++) {
   packet[u + i].payload = fec_encode(allpackets, i, PAYLOAD_SIZE);
   packet[u + i].seqnum = u + i;
}
```

The pseudocode generates f forward error correction packets from the u packets created in the previous steps. Each FEC packet is also assigned a sequence number. In the pseudocode, the first parameter to `fec_encode` is a pointer to all content packets from which the FEC packets are computed. The second parameter is the identity number of the FEC packet which is to be generated. The identity number is needed again by the FEC algorithm when the error correction packets are used for recreating lost packets.

At the end of this step the content of the original packets from the previous step will be as follows.

SeqNum, Encr(Payload | Padding)

The packets which were created by the FEC operation will have the following format.

SeqNum, FEC data

Source authentication

In step `IV`, hash trees are computed from the packets generated so far. The roots of the hash trees are then signed using a public key signature algorithm. Usually more than one hash trees are generated from the packets of one FEC block, because FEC blocks are typically wider than the authentication trees.

Figure 7 illustrates the creation of an authentication tree. As the first step in creating a hash tree, a cryptographic hash is computed over every w packets used to create a tree. The hash is computed over the entire packet, including the sequence number field and the possible padding in the packet. Also, the source and destination addresses which will be later added to the packet are included in the hash computation. That is the hash is computed as

hash = cryptohash(Dest | Src | SeqNum | Encr(Payload | Padding))

for packets generated from the data packets. For the packets generated from the FEC packets, the hash is computed as follows.

hash = cryptohash(Dest | Src | SeqNum | FEC data)

Next, a hash tree is computed from the generated cryptographic hashes. The hashes generated above directly from the packets are called the level 0 in the hash tree. The levels starting from number 1 in the tree are computed from the data in the previous level. The next level consist of hashes of the concatenation of hashes in the previous level. Each pair of hashes in the previous level are concatenated together and hashed to create the next level. That is, if the hashes $H_{0,0}$ and $H_{0,1}$ are concatenated together, the hash $H_{1,0}$ is its hash. Likewise, the hashes $H_{0,2}$ and $H_{0,3}$ are used to compute the hash $H_{1,1}$. New levels are created in the hash tree until only one hash is available to the algorithm. This hash is called the root of the hash tree. The root of the tree is then signed to create a root signature for the tree. The process is illustrated in Figure 7 for a tree 8 packets wide.

Several parallel authentication trees are built if one authentication tree is not wide enough to cover the entire FEC block. Each of the separate authentication trees will have a different root signature. The pseudocode below shows how an authentication tree is created.

```
int i;
int level = 1;
int level_width = w >> 1;

for (i=0; i<w; i++)
    H[0][i] = cryptohash_packet(packet[i]);

/* Construct the tree */
while (level_width > 0) {
  for (i=0; i<level_width; i++)
    H[level][i] = cat_cryptohash(H[level-1][i*2], H[level-1][i*2 + 1]);
  level++;
  level_width = level_width >> 1;
}

signature = crypto_sign(H[level-1][0], source_pk);
```

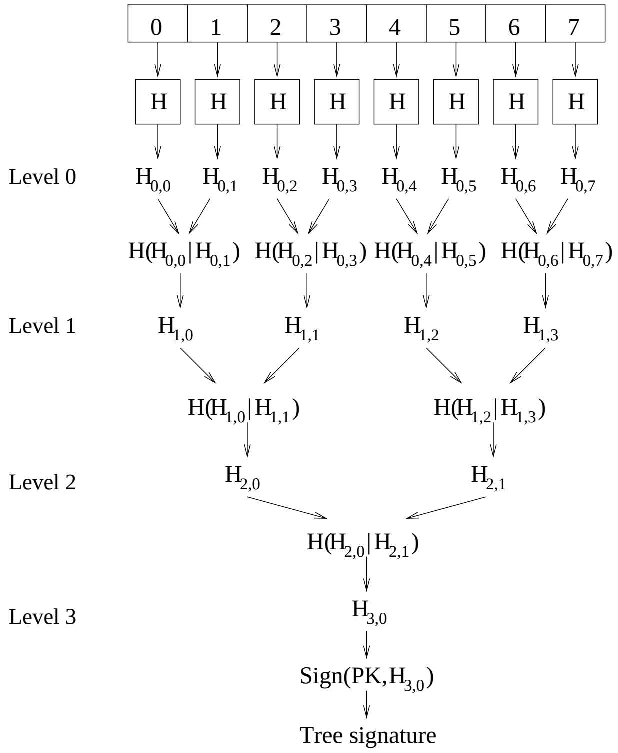

Figure 7: The authentication process.

Once the authentication tree has been created and signed, the required authentication hashes are added into the packets. The hashes which are added to a packet are the ones required to reconstruct the hash path from the hash of the packet to the root hash. That is, each packet needs to contain the minimum number of hashes from the hash tree, which can be used to reconstruct the hash path from the packet to the root of the tree. The pseudocode below performs this operation for each packet in the hash tree that was created above.

Figure 8 shows which hashes from the tree are added into the packet when constructing the hash path for packet number 5. This same example will be used again in Section 6.5 where the packet is authenticated using the data in the packet. The circled hashes and the root signature in the picture are added into the headers of the packet. For example, the packet number 5 from the tree in Figure 8 will include the hashes $H_{0,4}$, $H_{1,3}$, and $H_{2,0}$, in this order. This can be verified by running following pseudocode. The pseudocode chooses the hashes for all packets in the authentication tree. The hashes for packet number 5 are selected when the counter i has the value 5. The variable w, the width of the authentication tree, is assume to be 8 as is the figure.

```
int i, level;
char* packet_hashpath;

for (i=0; i<w; i++) {
   for (level=0; level<h; level++) {
      hashcolumn = (i >> level)^1;
```

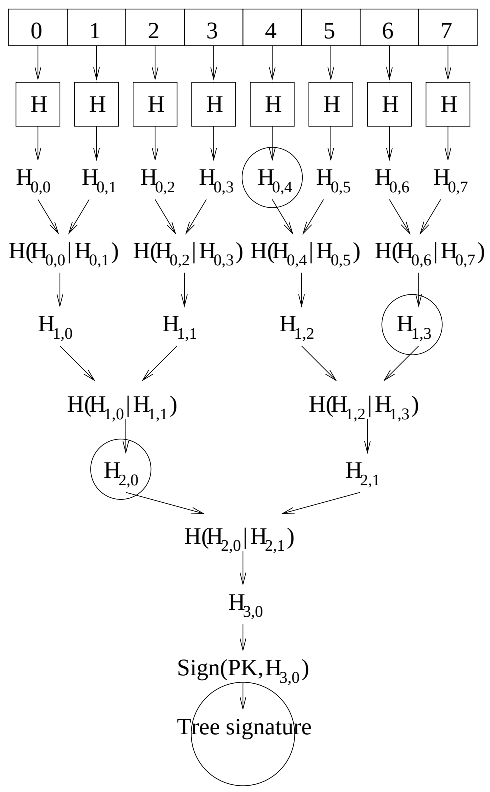

Figure 8: Example of source authenticating packet number 5 in the tree.

```
        packet_hashpath = H[level][hashcolumn];
        packet_hashpath += AUTH_HASH_LENGTH;
    }
}
```

At the end of this step, the content of each packet will be as follows.

PacketType, Hash-path, Signature, SeqNum, { Encr(Payload) | Padding / FEC data }

When the execution of the multicast packet creation reaches this point, one entire FEC block of multicast data has been created. The value for q is incremented by s and the process continues at step I as long as more content data is available. When the entire FEC-block of packets has been processed, the packets are moved into a transmission queue. From the queue, the packets are sent into the network.

6.5 Multicast cache

A multicast cache operates as a temporary storage between the multicast server and the terminals. A multicast cache listens to multicast transmissions from the network, stores the received packets to disk and resends data in response to requests from terminals.

In general, a multicast cache should only store multicast streams which are coming from a trusted source. Packets coming from trusted sources can be source authenticated using a public key from the content provider. Each content provider, whose multicast streams are to be stored by the cache, needs to give the verification key needed for the authentication to the cache operator. A multicast cache which receives multicast packets uses the verification key to ensure that the received multicast packets have been created by the legitimate multicast server. However, since the cache is only given the verification key, the cache cannot generate new packets for a multicast stream. Neither does the multicast cache require a key to decrypt the content delivered by the multicast streams. A multicast cache only operates on the packet level, and receives and resends packets.

Packet source authentication

Before a received packet can be stored, it needs to be source authenticated. A multicast cache needs to be preconfigured with information that binds multicast streams to verification keys which can be used to authenticate the data.

Each individual packet received by a cache is immediately authenticatable. The authentication of each packet can be done separately from every other packet. However, information received in earlier packets can often be used to make the authentication process much more efficient. The steps in source authenticating a received packet are the following.

1. A cryptographic hash is computed over the packet. The hash is computed over the source and destination addresses, the sequence number, the encrypted payload and the possible padding in the packet.

2. The hash path from the packet to the signature is reconstructed from the hashes included in the packets

3. The root hash is verified by checking the signature.

The hash path is reconstructed by the algorithm shown below in pseudocode. In the code `seqnum` is the sequence number from the received packet, and `nexthash` is a pointer to the first hash in the received packet. In the pseudocode, the variable `index` is initialized by taking the sequence number of the receive packet and by performing a bitwise `AND` with $w - 1$. This operation converts the sequence number from the packet into a index number in the packet tree. Note, that this works only when the variable w has been set to a value which is a power of 2, as required in Section 6.3. Next, the hash path from the packet to the root of the hash tree is regenerated based on the hashes in the packet. Two `cat_cryptohash` functions are needed, because the hashes need to be concatenated from either direction depending on the tree. For example, in Figure 9 hashes $H_{0,4}$ and $H_{0,5}$ are concatenated such that the hash from the packet is concatenated from the left, whereas the concatenation of hashes $H_{1,2}$ and $H_{1,3}$ receives the hash stored in the packet from the right. When the algorithm is completed, the root of the hash tree has been generated from the hashes in the packet.

```
int i;
int index = seqnum & (w - 1);
char *nexthash;
```

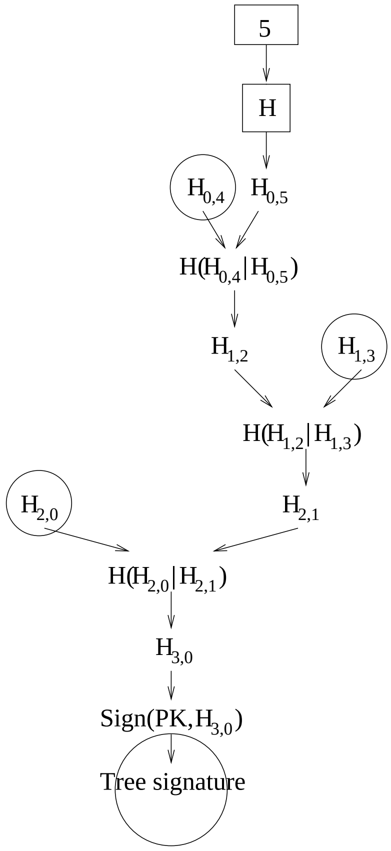

Figure 9: Example of receiver source authenticating packet with sequence number 5.

```
hash = cryptohash_packet(packet[index]);

for (i=0; i<h; i++) {
  if (index&1)
    hash = cat_cryptohash(nexthash, hash);
  else
    hash = cat_cryptohash(hash, nexthash);

  nexthash += HASH_LENGTH;
  index = index >> 1;
}
```

Figure 9 gives an example of the hash tree which is constructed to verify the authenticity of packet number 5 from this hash tree. The hash data stored in the packet is shown circled in the figure. After the hash path has been reconstructed from the data, the signature needs to be verified. The packet is verified in one of the following ways.

1. If the received packet is the first packet that is received from this authentication tree, the signature is verified using the verification function of the digital signature algorithm. In this case, if the packet passes

the verification, the root hash of the packet is stored in the memory of the cache along with the sequence number of the packet.

2. If the received packet belongs to an authentication tree from which at least one acceptable packet has been received, a much less computationally intensive verification path can be used. Since the packet belongs to an authentication tree from which another packet has already been received, a verified hash for the authentication tree already exists. In this case, the root hash computed from the received packet is compared with the previously accepted hash from the same hash tree. If the hashes match, the packet is accepted. On the other hand, the packet is dropped if the previously received hash and the hashes computed from the packet do not match.

A packet which passes the source authentication is stored in the cache to wait for retransmission.

6.6 Multicast client

The software which receives and stores the multicast data within the terminal is called the multicast client. The multicast client receives its configuration information from the user of the terminal and uses the information to selectively receive and store available multicast data.

The multicast client receives user selected multicast streams from the network. The received packets are authenticated using the algorithm previously presented for the multicast cache in Section 6.5. Packets that are lost from the multicast stream can be automatically requested from a multicast cache using a protocol described in Section 6.7. Received packets are stored in the memory of the terminal in encrypted form. This allows the client to receive packets without needing a decryption key while the data is being received. Also, the encrypted packets can be further delivered to another terminal in their original authenticatable form.

The multicast client communicates with application using a protocol built on top of the HTTP protocol. This allows the multicast client to be configured by a user using a graphical GUI through any existing web browser. The same protocol also allows the multicast client to deliver multicast data to most existing applications even if the application does not specifically support receiving data over multicast. The communication between the multicast client and the legacy applications is presented in Section 6.8.

6.7 Client-cache protocol

Like any application used for receiving multicast, the multicast client can listen to multicast transmission in real time. However, the number of packets that are lost from a multicast stream can be too large to be corrected by forward error correction if a terminal cannot listen to the multicast transmission all the time. One typical situation is when a mobile terminal temporarily moves into an area without wireless network coverage which is good enough to be used for receiving the stream. The client-cache protocol is used for requesting the lost packets after the terminal returns within network coverage.

The lost packets are retransmitted from a multicast cache. Essentially, the protocol can be used to compensate an unreliable link by using retransmissions.

Cache selection

A multicast cache periodically sends Cache Announcement messages over the network interfaces which are used to send multicast data to terminals. The goal of the Cache Announcement messages is to inform the terminals of the presence of a multicast cache which is willing to send retransmissions. One multicast cache can sent announcements over several interfaces, with possibly different network technologies, simultaneously.

The announcement messages are sent to a multicast address which is listened to by all multicast clients. Each Cache Announcement message contains at least the following information.

1. The contact address. The contact address is used by the multicast client as the destination address in the requests sent to this multicast cache.

2. A cookie value, which is a random number. The cookie value changes in every announcement message which is sent. When the client sends a request to the cache, it includes the most recent cookie value into the request. The value is used to ensure that only clients that are able to listen to the announcement messages can send acceptable requests to the multicast cache.

The cache announcement message can also include other information, such as the identity of the owner of the cache, a timestamp and pricing information. The announcement messages can also be signed using the multicast caches signature key, if such a key has been assigned to the cache. However, these extensions are outside the scope of this dissertation.

Based on the above specified information, the multicast client selects one or more of the simultaneously available multicast caches as the caches which will be used to request retransmissions. The client software can use variables such as available network capacity and pricing to select the multicast cache if several caches are simultaneously available.

Retransmission requests

Retransmission requests are sent by a multicast client when it detects that it has not received one or more packets that it needs to reconstruct the original content from the received packets.

The client software maintains a database of packets it has already received from each multicast stream. Every time the client software receives a packet from a multicast stream, it verifies whether or not it has received enough packets from every FEC block that has already been sent by the server. If the multicast client detects that it has not received a sufficient number of packets from each sent FEC block, it can send a retransmission request to one of its currently available multicast caches. Typically, the client software requests the packets that are needed most urgently. That is, the client attempts to reconstruct the incomplete FEC block with the smallest sequence number first.

The sequence numbers of the packets missing from the client software are carried in the request message to the multicast cache. The message body consists of pair of numbers. The first number of the pair gives the sequence number of the first packet. The second number of the pair gives the number of packets missing after the sequence number given in the first number of the pair. The request message also carries the source and destination addresses of the stream from which packets are begin requested. The most recently received cookie value is also included in the packet.

$$\textit{Dest addr, Source addr, cookie value, } \underbrace{\left\{\textit{seq. start, num. packets}\right\}}_{\textit{Possibly repeated part}}$$

An example of a request message is given below:

FF1E::1234, 3FFE::1, 0x1A6F8BB2, 102, 5, 115, 4

In the example above, the client software requests packets in the sequence number range of 102-106 and 115-118 to be retransmitted.

Retransmissions

The retransmission requests are sent to one of the available multicast caches. After receiving a request, a multicast cache first verifies that the cookie value received in the packet matches one of the cookies that have been transmitted by this cache within the last few seconds.

Next, the packet is compared with request messages received from other hosts. If multiple requests for the same packets are received from separate nodes, the requests for duplicate packets are removed from the request. Since terminals that are sending requests do not know what requests have been sent by other nodes, this comparison is needed to prevent multiple retransmissions of the same packet within a short period of time. Then the request is entered into the retransmission queue to wait for an opportunity to be sent.

When the radio interface is available for sending retransmission packets, a retransmission request is taken from the queue. Next, the multicast cache needs to decide which packet to retransmit in response to the request. Naturally, the retransmitted packet needs to be from the multicast stream identified by the request. However, due to the use of FEC, the sequence number of the retransmitted packet does not need to match exactly to the requested sequence number. Any packet with the following properties are suitable to be retransmitted.

1. The transmitted packet is from the same FEC block as the requested packet.

2. The client has not already received the packet from this stream with this sequence number.

That is, any packet, from the correct FEC block, unreceived by the client is suitable for replacing a lost packet. However, it is usually impossible for the multicast caches to know which exact packets have already been received by the clients. This makes the second property of the above list difficult to utilize. The most simple strategy is to send exactly the packet which was requested in the message. This method is suitable for the situation where only one terminal is requesting packets from this FEC block to be sent.

Another strategy may be more appropriate when sending retransmissions from one multicast stream simultaneously to multiple terminals. If the above strategy is used in this case, each client needs to be retransmitted repair packets individually, since the set of lost packets will most likely be different in every host. In this case, the goal is to send packets which are most likely unreceived at all of the terminals. That is, the multicast cache ignores the exact sequence numbers in the requests and retransmits, for example, only FEC packets from the specified FEC block.

6.8 Client-application communication

The architecture needs to be able to communicate with existing applications. Since most existing applications are unable to receive multicast data directly, an adapter is needed for communication between the user and the multicast client application. This interface is used for configuring the multicast client, for example, to select what content is received from the network. The natural communication interface between the multicast client and the client application is the protocol stack and the socket API. The client application implements a web server which listens to connection only from the local host. The user can use any web browser for configuring the multicast client application. Commands are sent to the client application as URLs. The responses form the client application are shown as web pages in the browser.

Another interface is also needed between the multicast client application and the helper applications or plugins which are used to display the content to the user. This interface is needed to transfer content from the client application to the helper application or plugin which is used to display the content. This interface can also be implemented through a local web server and the socket API. Figure 10 illustrates the communication paths between the components in the terminal. Path 1 is used for configuring the client software, while path 2 is used for delivering content to the application used for displaying the content. At path number 3, the content is delivered from the player software to the output devices. The output devices can be, for example, a video device and a sound card.

Figure 11 gives an example of how the data is delivered from the client software to the MPlayer software. The browser in the example can be any browser capable of using helper applications or plugins. We assume that the client application has already received enough content from the multicast stream to start displaying it, although the content may be only partially available in the terminal. The numbers of the steps are shown in the figure.

In step 1, the user clicks on a button in the web browser. The button is used for starting the player application. In response, the web browser downloads a playlist file from the web server. The playlist file contains the URL which can be used for retrieving the content from the local web server.

In step 2, the browser selects the appropriate helper application for the playlist file that was received from the web server. In this example, the playlist file downloaded is configured to start the MPlayer application.

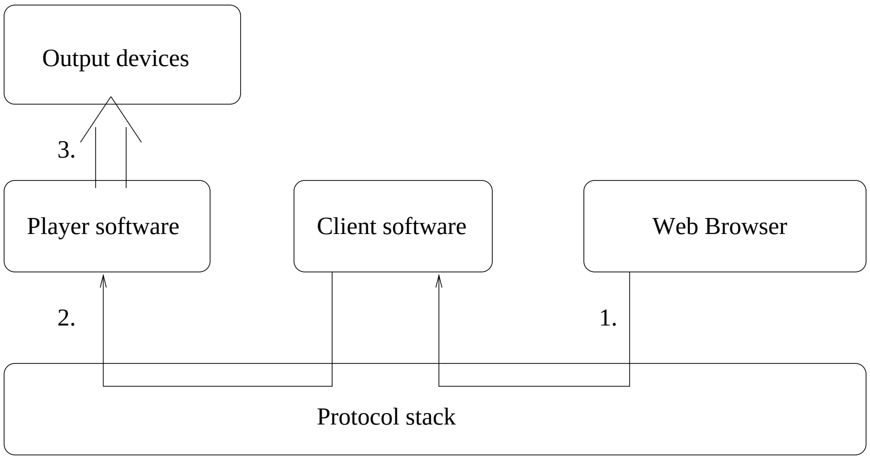

Figure 10: Legacy application communication.

In step 3, the playlist is given as a parameter to the helper application. Next, MPlayer opens the playlist file, which contains a URL pointing to the local host's server and to the content chosen by the user. A connection is opened from MPlayer to the web server within the terminal, and the content is downloaded. Finally, the content is played on the output devices.

6.9 Control of content

Content is usually delivered encrypted over the architecture. Instead of selling content, the content provider sells keys which are needed to consume the content. Since content can be delivered over multicast, the key used to decrypt the content is necessarily the same for all users of the content. Therefore, the architecture must include a system for protecting the keys and the decrypted content from being exposed to a user who could remove the protections from the content.

The content control in the system consists of two parts. The first part, the security module, is built into each terminal used to receive content. It is used for securely receiving keys from the key server. The second part, the keyserver, is a server in the network. It is usually operated by the content provider. It supplies keys to authorized security modules.

Security module

The security module handles key management and content protection in the terminals. The goal of the security module is to prevent unprotected content from being extracted from the terminal. Minimally, each security module is preconfigured with the following information before being installed into a terminal.

1. The identity of the security module. The identity consists of a of a public key and the corresponding private key.

2. A certificate which verifies the validity of the identity. The certificate is signed by a TTP.

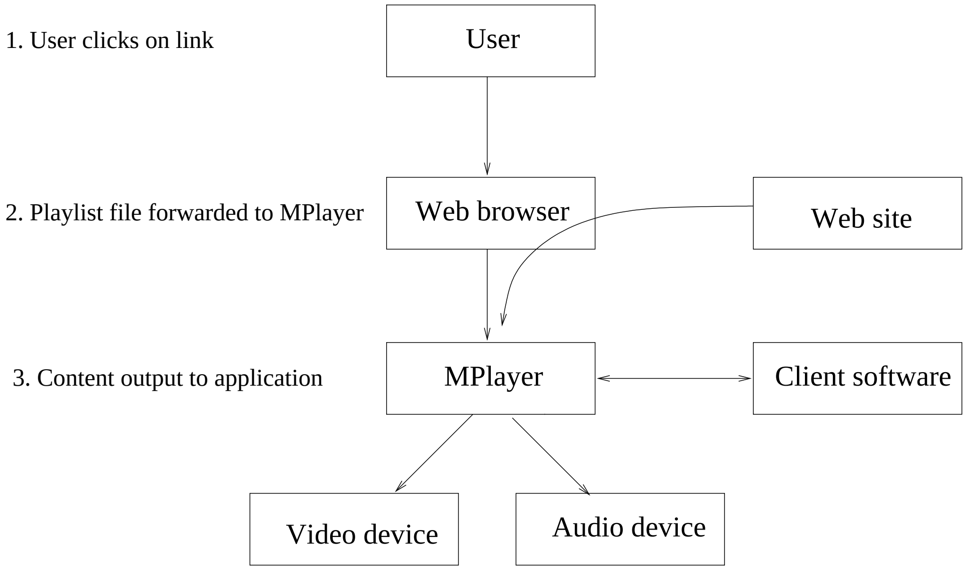

Figure 11: Example of client-application communication.

The identity and the certificate are used in requesting keys from the key server. The security module can also implement other functionality including watermarking and re-encoding prevention. These security optimizations are discussed further in Section 8.2.

The security module can be implemented either in hardware or in software. A tamper proof hardware module may be required if the level of protection needed is high. In an environment where only a low level of security is required, the security module can be implemented purely in software.

Key management

Content is delivered encrypted to the terminal. When the user decides to use the content, the security module connects to the key server to retrieve the key. The request identifies the content for which the key is required. In the request, the security module also includes its public key and the certificate which proves its validity.

When the key server receives a request, it verifies that the requester is authorized to use the content. This verification is done, for example, through a certificate issued by a content distributor. When the authorization has been verified, the authenticity of the security module's identity key is verified. If the request passes all checks, the key server encrypts the content key with the public key received in the request. The encrypted key is then sent to the security module. Now, the security module has received the key it needs to decrypt the content. The security module can start feeding the content to the application over the local socket connection.

7 ANALYSIS

This chapter analyzes our solution in Chapter 6 against the criteria presented in Chapter 3.

7.1 Technology independence

The architecture should allow operation over any existing and future network technologies which can be used to transmit IP packets. This property gives the architecture a solid foundation, which will enable it to operate without modification even with network technologies that have not yet been designed.

Network technologies

One of the requirements in creating the architecture was that the architecture should operate over any wired and wireless network technology, whether the technology is bidirectional or unidirectional. Since only the basic properties offered by all IPv6 capable networks were used in the design, the architecture can be used over any network technology capable of delivering IPv6 packets.[10]

However, some network technologies can perform suboptimally when used to deliver multicast data. Figure 12 illustrates the problem for IEEE 802.11g WLAN. In the figure, two nodes are listening to the multicast transmission from the access point. The two remaining nodes are not interested in receiving the packets. Both of the multicast listeners are located within good radio coverage of the access point, and they can communicate with the access point at the maximum 54Mbps capacity. The two remaining nodes are further away from the access point, and they are able to communicate only at a considerably lower capacity.

In the IEEE 802.11 protocols, receivers do not send any acknowledgments in response to receiving multicast packets. To send multicast packets, the access point must use one of its low capacity modulations to ensure that the packet can be received by all of the nodes in the area. That is, the access point may be forced to utilize the 2Mbps modulation to ensure that the transmission can be received fairly reliably even by the more distant nodes. In the case presented in Figure 12, a more optimal solution could be to send the content directly to the two receivers over unicast at 54Mbps. However, if the number of multicast listeners in the example were larger or located further away from the access point, multicast transmission would again be the preferred alternative. If the access point does not know the number of receivers that are receiving the multicast transmission, it can be difficult to determine when to switch to unicast transmission.

Another network technology where unicast is preferable over multicast is GPRS. Even if GPRS technically does support multicast, it is currently implemented by sending each terminal the same packet over separate unicast connections. Thus, all the advantages of multicast are lost.

[10]Alternatively, IPv4 can also be used.

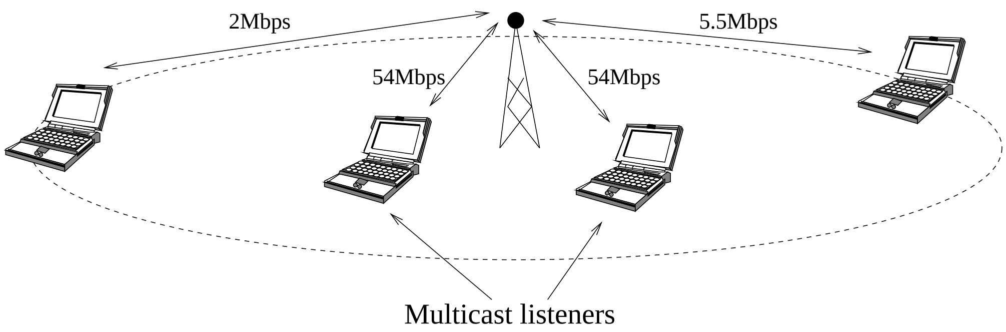

Figure 12: Multicast over IEEE 802.11 protocols.

Heterogenous wireless access

The architecture allows the terminal listening to the multicast stream to always select the best alternative from the currently available access technologies. The terminal can choose the technology, for example, based on available capacity, cost, network coverage, or reliability of the technology. An architecture which can be used for selecting a suitable access technology is presented in [17].

The architecture also allows the content to be reliably reconstructed from packets which were received from several different access networks. The multicast caches that are used for retransmissions do not need to be trusted by the terminals.

The architecture does not assume any special abilities from the network technologies besides the ability to transmit IP packets. However, in some cases it is possible, that the implementation of the network technology may be suboptimal, for example, for using multicast delivery. For example, in the above case of IEEE 802.11g, using unicast instead of multicast can in some cases be preferable in terms of efficiency. The effects of specific network technologies, in some cases, need to be considered when implementing the architecture.

Incomplete network coverage

The architecture is able to operate in a network with incomplete coverage. The three techniques used by the architecture against network outages of arbitrary length are forward error correction, retransmissions and buffering.

Forward error correction and retransmissions ensure that no packets are lost from content even if the mobile terminal leaves the coverage area of the network. The packets that are lost during the network outage can be retransmitted from a multicast cache once the terminal return to an area with network connectivity. The effectiveness of FEC and retransmissions are further discussed in Section 7.4.

Buffering is another method which can be used by the terminal for hiding temporary network outages. When the terminal is used to display, for example, real time television, the terminal can temporarily store the received content in a buffer. If the network coverage is temporarily interrupted, the terminal can continue to display content from the buffer. As soon as the connectivity to the network is restored, the terminal can request retransmission

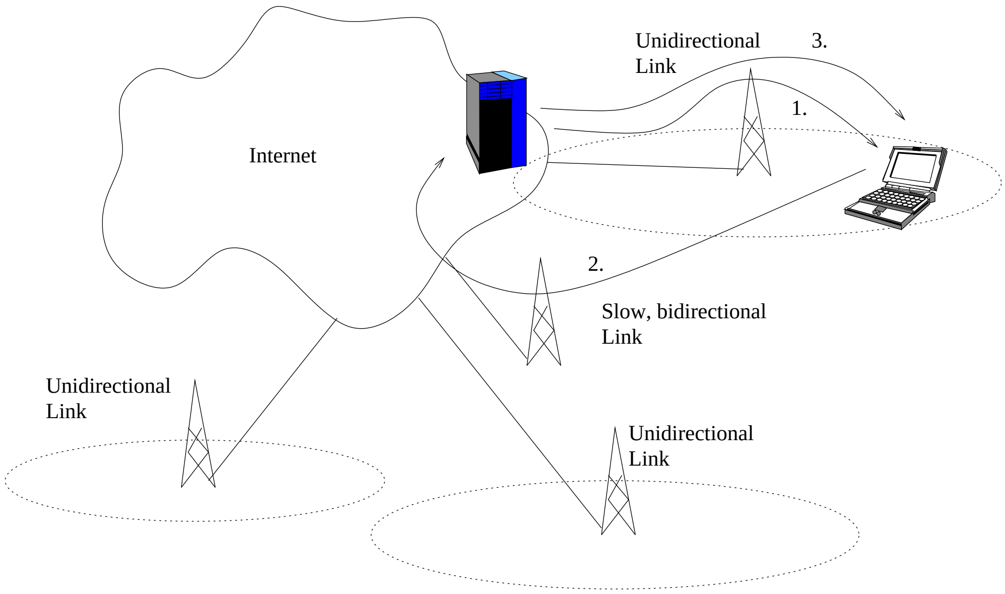

Figure 13: Operation over unidirectional links.

of the lost packets from a local multicast cache. The cache is then used to refill the buffer that was used during the outage. To work, buffering requires content to be shown a few seconds behind real time, even in, for example, the case of live television. The amount of delay needed is equal to the maximum time of network loss which needs to be tolerated without break in the display of content. If the maximum time of loss of network connectivity in a short period of time is smaller than the delay, the user experiences no break in the content.

Unidirectional links

The architecture allows retransmission to be delivered over access networks consisting of unidirectional last-hop links. The cache announcement messages sent by multicast caches contain information which tells the terminals which address can be used for contacting the server which sent the announcement. If the wireless access technology is unidirectional, the request message to the multicast cache can be sent over some other wireless network technology. One example of a networking technology for sending requests is GPRS. While GPRS has low throughput, the technology is available practically everywhere, and the capacity needed for sending requests is very small.

Figure 13 illustrates the operation of the retransmission protocol over unidirectional last-hop links such as DVB-T or DVB-H. In step 1, the terminal receives a cache announcement message from the multicast cache over the unidirectional link. The message contains the contact address where the multicast cache can be reached. In step 2, the mobile terminal sends a request message through the low capacity bidirectional technology, such as GPRS. Finally, in step 3, the multicast cache sends the requested packets to the mobile terminal.

7.2 Business feasibility

The architecture is only useful if it can be successfully deployed in the real world. We discuss the ways the architecture uses to make transitioning content delivery to the architecture as simple as possible for both users and businesses.

Implementability
A proof of concept implementation of the architecture has been built for Linux. The size of the implementation is roughly 6000 lines of code, and it implements the server, cache and client functionality of the architecture. The source code can be downloaded from
`http://www.tcs.hut.fi/Software/Multicast`.

Legacy applications
Taking the architecture quickly into use requires that it is able to operate with existing applications. This requirement is especially important within the terminal devices as they are operated by the users, and installing new or updating applications which are not absolutely necessary would be an unnecessary and complex task.

Our implementation of the architecture communicates with existing applications using the HTTP protocol. The protocol is used for two purposes by the architecture.

1. The architecture allows the user to configure the client software using a standard web browser such as Mozilla or Opera. The user can choose, for example, what content is downloaded.

2. The architecture uses HTTP to deliver content to plugins and helper applications. The received content is distributed to client applications from a local HTTP server. The applications open a network connection to the local server. Any legacy application which is able to receive data from a HTTP stream can be used to display the content. Alternatively, the architecture can save the content on disk and give the file as a command line parameter to the application or plugin.

Our implementation has been successfully tested with the Mozilla and Opera browsers. The helper applications used include MPlayer and gv.

Data transfer vs. data use
The architecture allows transferring content to the terminal without committing to getting keys for the data. That is, the user can transfer content to the terminal in encrypted format whenever ample network capacity is available. The keys for using the content need to be received only once the user decides to use the content.

Content can be received from the network whenever it is the most convenient for users. Data can be received either from a wireless network or from a wired network using any existing access technology. Example use cases include users downloading every newspaper available at a newspaper stand. If the content is transferred over a wired link or over a short range high capacity wireless connection, the cost of transmission is virtually zero. Additionally, the time required for transferring even tens of megabytes of content can be only a few seconds. The keys to the content can be requested later over any network technology. The size of the key information is so small that, for example, GPRS can be used for transferring it without large delays or costs.

Illegal copying

The architecture has been designed with several technical methods of discouraging unauthorized use and distribution of content.

1. Making a copy of the unencrypted content has been made as difficult as possible. The decryption key to remove the encryption can be protected from being leaked to the user. The keys are managed by the security module which never gives the key out in plaintext. If strong security is required, the security module can be implemented as tamper proof hardware.

2. All content that is given out from the security module is digitally watermarked. The watermark gives the identity of the security module used for decrypting the content. The watermark data can then be used to track down the user if an illegal copy with encryption removed is being distributed.

3. The quality of the copied content can be degraded by adding information into the content in the security module. The additional information can be inaudible or invisible, but can make the content more difficult to re-encode. If the content is re-encoded, for example by capturing the video from the display device, the resulting content will have decreased quality or significantly larger size.

All of the techniques in the above list are optional. The content providers and distributors can only choose the techniques which they deem to be sufficient.

Encryption

Since the content is protected by encrypting it, the architecture must prevent the content keys from being leaked to attackers. If the attackers are able to extract the content key, the content can be decrypted into plaintext which can be used without authorization in any terminal. The strength of the protection is depending on the implementation of the security module. A security module in software can implement the same functionality as a hardware security module. However, a software security module is necessarily less secure than a security module properly implemented in hardware.

From the point of view of the content provider, a security module implemented in hardware is preferred. However, the hardware implementation is likely to result in more expensive terminals. If a content provider believes that its content is too valuable to risk illegal copies, the content provider can refuse to sell content for use in terminals with a software security module. Another possibility is to sell content at a cheaper price to users with hardware security modules.

Watermarking

All content given out from the security module should always be digitally watermarked. While the watermarks should be completely invisible to the user of the content, it should be as difficult as possible to remove from the content. The security module that was used for decrypting the content can be identified from the watermark. Since an illegal copy can always be traced back to one specific security module, the system discourages users from making illegal copies.

Even if we assume that the watermark cannot be removed from the content, the system is still not always effective. Identifying the security module used for decoding content is hardly enough to identify the human using the module. First of all, a security module can be purchased using a false identity. Second, even if a security module was purchased with a real identity, the user can always claim that the device was stolen. In both cases, the watermark can be used for identifying the security module used for decrypting the content.

Watermarking is mostly a psychological method against distribution of illegal copies. It can be an effective deterrent against a person who is about to make a copy of a movie and to give it to friends. On the other hand, watermarking has little effect on professional pirates who can use a security module that has been obtained anonymously.

Another problem in watermarks is that all content cannot be watermarked. Watermarking can be used effectively, for example, in movies, music, and pictures. However, it cannot be used, for example, for plain ASCII text.

Re-encoding prevention

The security module can output content in a format which is processed as far as possible. If the module is used for protecting movies, the output from the security module can be bitmaps which are ready to be transferred into the memory of the video card. If a user intercepts the data flow between the security module and the output device, the user needs to re-encode the content before it can be illegally distributed. The security module can then add extra information into the frames to make the data difficult and inefficient to re-encode. The result of this system is an illegal copy which has either poor quality or large size in comparison with the original legal and encrypted content.

To be efficient in preventing illegal copying, the system needs to be efficient enough that the ratio between quality and size of illegal copies is sufficiently smaller than with legal copies. If the difference is large enough and a legal copy of the content is affordable enough, users will probably be persuaded into purchasing a legal copy of the content.

7.3 Mobility

The architecture must allow the mobile terminal to switch from one access point or access technology to another.

Mobility between access points

The delay in switching a mobile terminal from one access point to another depends on several factors such as:

- Finding a new link. If the used wireless network technology does not enable the mobile terminal to search for new access points while communicating with the old access point, the connection to the old access point must be terminated before finding a new access point.

- Acquiring information the information needed to operate under the new access point. This information includes, for example, the new address for the mobile terminal.

Additional delay may result from the inability to immediately connect another access point, for example, due to moving temporarily outside the network coverage. During this delay, the mobile terminal cannot receive packets from the network. Packets that are transmitted during the access point change are lost.

The mobile terminal can use its retransmission protocol to request the lost packets to be sent again from the nearest multicast cache. Alternatively, if the network outage during the access point change is short enough, the mobile node may be able to regenerate the lost packets using the FEC data in the multicast stream.

Data reconstruction

In the architecture, parts of content can be received from any number of access points and multicast caches. Every packet can be source authenticated using the information received in the packets. The multicast caches and access networks that are used for delivering the data do not need to be trusted.

7.4 Reliability Analysis

The architecture is used for transferring data over an unreliable packet switched network. Furthermore, mobile terminals may occasionally move outside the coverage of wireless networks. The architecture must be able to deliver the content reliably even in such an environment.

Ordinary multicast

Multicast delivery of packets over the Internet gives no guarantees about packets being delivered to their destination. The delivery of multicast over IPv4 or IPv6 is purely a best effort service. That is, any packet in a multicast stream may be dropped within the network. Applications wishing to use the multicast delivery need to provide adequate reliability in one way or another. Traditionally, the service to provide reliability has been built into each application separately.

Packet loss

In *random packet loss*, only one or a few consecutive packets in a stream are lost. This type of packet loss can be caused, for example, by an unreliable wireless link or by congestion in the network.

In *burst packet loss*, the connection from the sender to the receiver is broken for a longer period of time, and the number of unreceived packets can be arbitrarily high. The typical reason for this is, for example, a mobile terminal which has temporarily moved into an area without network coverage.

FEC

Forward error correction is an efficient method against random packet loss. Even if a small percentage of packets are dropped in the network, the content can still be reconstructed by the receiver as long as a sufficient portion of the packets are delivered. The repair operation is completely invisible to the sender since no packets need to be retransmitted. Furthermore, the technique scales limitlessly as it does not require any feedback to be sent to the server.

However, FEC is not always sufficient for repairing content from burst packet loss. The amount of redundancy added to the stream determines the maximum amount of packet loss which is tolerated by FEC. Once the maximum packet loss rate is exceeded, FEC alone can no longer be used for repairing the content. For example, if a mobile terminal temporarily moves to an area without network connectivity, the resulting network outage can be arbitrarily long. In such a case, no amount of FEC data can be guaranteed to be enough for repairing the data. Furthermore, the amount of FEC data in a packet stream should be limited to an amount which does not unnecessarily congest the network.

While FEC is an efficient protection against random packet loss, it alone is unsuitable for recovering from all types of packet loss. In the general case, retransmissions are required to recover from burst packet loss.

Retransmissions

Retransmissions are another solution to packet loss. A retransmission requires the following steps in our architecture.

1. The receiver detects that pieces of data have been lost, and it cannot be regenerated by FEC techniques.

2. The receiver creates a retransmission request and sends it to a multicast cache.

3. The multicast cache sends more data packets to the receivers. The receovers can use these packets to recover from the missing data.

The major benefit of retransmissions over FEC techniques is reliability. As long as multicast caches have received the required number of packets from each FEC block, the required packets can eventually be transmitted to the terminals. The retransmission operation does not necessarily succeed on the first try, but the operation can be repeated as long as is necessary.

However, retransmissions alone are not a suitable solution against random packet loss. As the above steps illustrate, in our architecture each retransmission requires the terminal to send a request message to the multicast cache. The cache then retransmits the lost packets. If FEC were not used, each terminal would separately need to request the retransmission of every packet

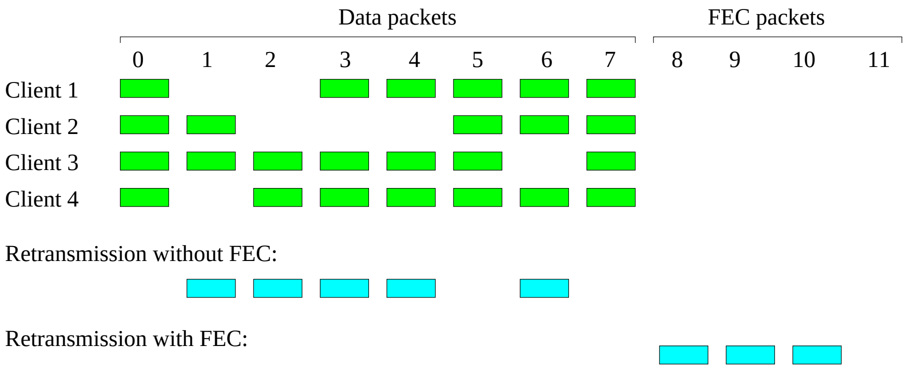

Figure 14: Example of the efficiency of retransmission using forward error correction.

it has missed. The multicast would then retransmit exactly the packet which were requested by the terminals. Additionally, each terminal is likely to lose a different set of packets than other terminals in the same area. If the number of terminals in an area is large, each packet would probably be lost by at least one terminal in the area, even if the number of packets lost by each terminal is small. Thus, the multicast client would need to retransmit virtually all packets again to repair the packet loss.

When FEC is used for retransmissions, the situation becomes much simpler. In this case, the multicast cache only needs to retransmit the maximum number of packets missing from any one terminal. If the retransmitted packets are FEC packets instead of the same packet that were already sent, all of the terminals can benefit from the retransmission.

Figure 14 illustrates the solution. In the figure, 4 clients are receiving 8 data packets. Each client has received some subset of all of the packets. For example, client 1 has lost packets number 1 and 2, whereas client 2 has lost packet number 2, 3, and 4. Only the packets 0, 5 and 8 have been received by all of the clients shown in the figure. If FEC is not used, the multicast cache would need to retransmit all of the packets, except the packets numbered 0, 5 and 7. However, if FEC is used, only three FEC packets needs to be retransmitted.

8 SECURITY ANALYSIS

The system needs to be able to operate even under attack by hostile nodes. This criterion can be divided into the subcriteria of content integrity, content confidentiality, and availability. More informally, the system needs to ensure that the receivers are delivered content that has been created by a known content provider, and the information is received without having been modified during transfer. Additionally, the content must be usable only to an authorized user, and no node in the network should be able to prevent the delivery of the content to authorized users.

8.1 Content integrity

In the system, only a server with the correct signature key is able to create content which is delivered through the architecture. The verification key, on the other hand, is available to every cache and receiver in the network. Every packet delivered through the system can be source authenticated before being processed by a node of the system. Any packet which fails verification is immediately dropped.

Since every packet delivered by the system is verifiable independently from other packets, the system has the following properties.

- Only packets signed by legitimate signature keys are delivered. Packets without signatures or with incorrect signatures can be dropped without being delivered to the next hop.
- Since the signatures are computed over the payload, a receiving node can verify that the content carried by the packet has not been modified while being transferred in the network.
- Any node in the network can verify that the content has been created by the sender who claims to have created it.
- A receiver can safely request packets from even untrusted caches, since the multicast cache nodes are unable to create forged packets.
- Any node in the network with knowledge of the authentication key is able to filter out forged packets from a packet flow consisting of both legitimate and forged packets.

Tree authentication

Optimally, each packet delivered by the system would carry a separate signature computed over only itself. However, both the generation and verification of sufficiently strong signatures is a very computationally intensive process. The generation of signatures would certainly be possible if a computer powerful enough is allocated as the server. However, the problem lies in the verification of packet generated by such a system. Low energy consumption is an important design criterion for most mobile terminals. Such devices will necessarily have processors which are not capable of verifying high capacity packet streams if each packet is signed separately.

The solution used by our architecture is based on hash trees. The goal is to combine the best properties of signing individual packets with less computational overhead. In the hash tree based algorithm, one signature can be used to sign several packets at the cost of adding some extra authentication data into each packet.

The algorithm for computing the hash tree was described in Section 6.4. The value of the root hash depends on every packet used to compute the hash tree. If any packet is modified, all hashes which are computed based on this packet also changes. Since the root hash depends on all packets in the hash tree, if any one of the packets in the tree are modified, the new root hash computed from those packets will differ from the previously generated root hash. The details and the analysis of the hash tree authentication algorithm can be found in [34].

Authentication efficiency

The selection of parameters for the authentication tree computation is a tradeoff between the size of headers in packets, the strength of the authentication, and the amount of processing capacity needed.

The amount of processor capacity needed is decided by the choice of signature algorithm, the strength of the key, and the width of the authentication tree. The amount of computation needed for verifying the authentication is dominated by the time needed for verification of the signatures. Therefore, it is vital to minimize the number signatures that need to be verified by the receiver. However, increasing the number of packets in an authentication tree also increases the number of hashes needed in each packet. The second important factor in adjusting processor capacity requirements is the choice of signature algorithm. A signature algorithm with low cost verification is preferred. The signature creation algorithm does not necessarily need to be computationally efficient, since the server is usually running in a computer with ample processing power.

The size of headers in the packets is another important consideration when choosing parameters. If each packet is authenticated by an individual signature, no hashes are required in the packets. However, whenever the number of packets in an authentication tree is doubled, one extra hash is required in each packet. For example, an authentication tree 64 packets wide requires a total of 6 hashes in each packet. The size of the resulting header also depends on the size of individual hashes which are included in the packets. The two most widely used cryptographic hash algorithms are the MD5 and the SHA1 algorithms. The hashes created by the MD5 algorithm are 16 bytes long while the hashes created by the SHA1 algorithm are 20 bytes long.

The strength of the authentication is the final parameter requiring adjustment. The strength is decided by choosing a signature algorithm, the key length for the signature algorithm, and by choosing the length of the hashes used for computing the hash tree. The lengths of the hashes and the length of the signatures should be selected in a way which makes creating forged packets sufficiently difficult to be a significant threat.

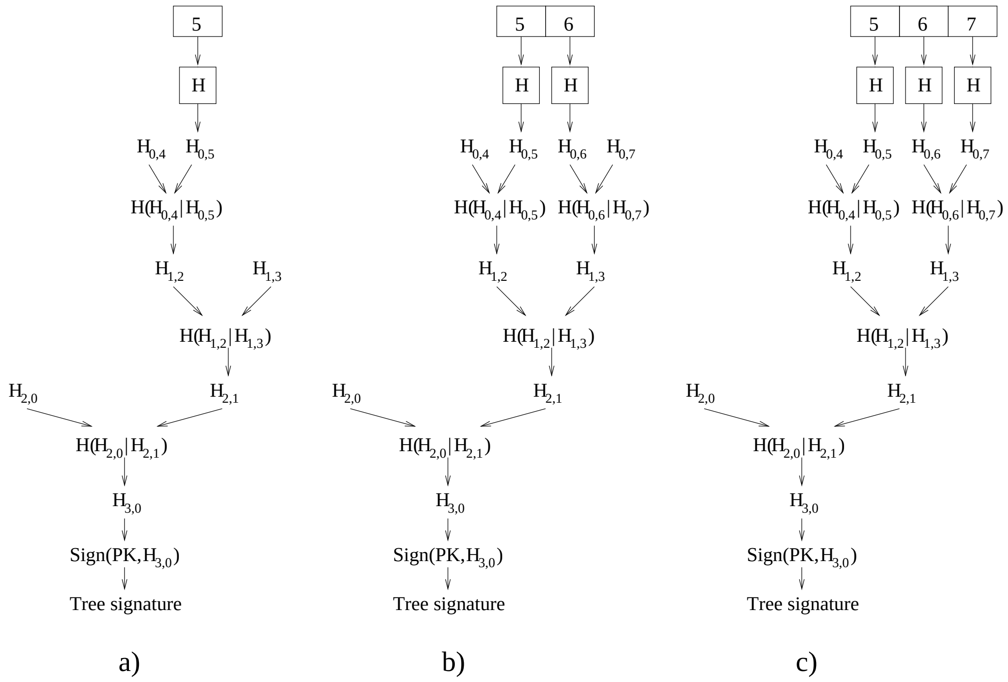

Figure 15: Example of optimization for the tree authentication algorithm.

Optimizations for authentication

The most simplified implementation of the authentication algorithm only stores the root hash from each authentication tree. The stored root hash is then compared with root hashes computed from other packets received from the same authentication tree.

The authentication algorithm can be optimized by storing every hash created and received in the authentication process, instead of only storing the root hash. When the next packet is received from the same authentication tree, the entire hash path does not need to be computed. Instead, only the part of the hash path which has not been generated by an earlier packet needs to be generated. In the optimized algorithm, the hash path is generated only until the path converges with the already generated hash tree.

Figure 15 illustrates the authentication optimization through an example. Figure 15a shows the authentication path which is generated when the packet number 5 from the tree is received. When using the optimized version of the algorithm, the entire hash path is stored in memory. All of the hashes shown in Figure 15a are stored. When packet number 6 from the same authentication tree is received in Figure 15b, the algorithm computes the hash $H_{0,6}$ from the received packet. The hash $H_{0,7}$ is received in the packet. The hash $H_{1,3}$ is computed by hashing the concatenation of the two hashes. The resulting hash $H_{1,3}$ is compared with the same hash which was received in packet number 5. If the hashes are equal the packet is accepted and the new hashes which were computed when verifying packet 6 are stored. Instead of regenerating the entire path to the root node, only the path from the new packet to the already generated authentication tree needs to be computed. Next, when packet number 7 is received in Figure 15c, the hash $H_{0,7}$ has already been received earlier with packet number 6. The newly received packet is hashed, and the hash from the packet is compared with the hash which was received in an earlier packet. If the hashes are equal, the packet is accepted.

The optimization allows packets to be authenticated without using all of the hashes in the packets. In principle this could be used for reducing the number of hashes included in the packets, if it were possible to predict in which order the packets are delivered and which packets are dropped. Unfortunately, since we assume that the algorithm is being used over an unreliable packet switched network, neither of the assumptions hold. Even if the optimization allows using fewer hashes from the packets for authentication, it does not help in reducing the number of hashes required in each packet.

The optimization can be used to minimize the number of hash operations needed to verify the authenticity of the packets. However, as the time required for computing hashes is small and the optimization requires additional memory to be used, the benefit from using the optimization is not clear.

8.2 Content confidentiality

The architecture can be used to distribute content either in encrypted format or in plaintext. The goal of encrypting the content is to protect the content provider's rights against illegal copying and distribution. The goals are fulfilled by designing the following properties into the system.

1. Authorization from the content provider or one of the content providers distributors is needed to use the content.

2. The content can only be used on a system which verifies the user's right to use the content before allowing access to the content.

3. It is difficult to make illegal copies where the content protections have been removed and the content quality remains good.

4. Illegal copies of the content can be used to identity the device which decrypted the content.

Distribution of content is not restricted in any way. On the contrary, content providers can encourage users to distribute content to other users. However, before the received content can be used, the user needs to retrieve a decryption key from the key server.

Since the content needs to be deliverable over multicast, every user of the system receives the content encrypted using the same key. However, since knowledge of the key needed for decrypting the data would allow a user to remove the encryption of from the content, the key must never be given to any user or piece of software which might reveal it.

When a key server receives a request from a terminal, it first verifies that the terminal is authorized to receive a key for the content. The verification may be done in any way. After verification, the key server encrypts the content key using the public key known to belong to the terminal which requested the key. The encrypted content key is then sent to the terminal over the network. The encrypted content key is given to the security module, which uses its private key to decrypt the content key.

The content key is encrypted before being sent from the key server, and it is decrypted only once it is within the security module. The content key is never available in plaintext while being delivered over the network. The content key is available in plaintext only to the multicast server, the key server and to the security module in terminals. Since the multicast server and the key server are operated by the content provider, the security of the keys within these servers can be ensured relatively easily. However, the problem lies within the security module. Since the content key needs to be stored in a device which is physically available to potential attackers, the security module itself needs to be built in a way which makes it sufficiently difficult to extract the key from it. Every security module has in its possession two kinds of keys which it needs to keep secret. The keys are content keys and the identity key of the security module.

The content key is used to decrypt one specific piece of content, for example a movie. If a content key is revealed, the content encrypted with the key can be decrypted. Decryption of the content allows any user in the network to use the content in any device without paying the content provider for the authorization.

The other key which can be compromised is the private key part of the identity key of the security module. This key is used to decrypt the content key within the security module, and the content key can be seen as a master key. Since the identity key is used for protecting the content key while it is sent to the security module, the identity key can be used to reveal any number of content keys. An attacker who has been able to extract the private part of the identity key can perform a very simple and efficient attack. The attacker legitimately buys the authorization from the key server. The key server then sends the content key to the security module. The attacker then intercepts the data containing the encrypted content key and uses the private part of the identity key to decrypt the content key. The content key can then be used to remove any protections from the content. However, the key is restricted to decrypting only the one content which encrypted using it.

The above attacks demonstrate the importance of the security module's ability to maintain the keys in its possession secret from any attackers. Especially the revelation of the private part of the identity key makes the content providers unable to prevent good quality illegal copies from being distributed.

The security module can be implemented in either software or hardware, depending on the value of the content being protected.

Software security module

In this alternative, illustrated in Figure 16, the security module is implemented completely in software. The software implementation of the security module should be written in a way which makes it as difficult as possible to extract the keys from within the module. Methods which can be used for securing the security module include code obfuscation tools and any other methods which make reverse engineering the implementation more difficult.

However, no matter how well the implementation is protected, the decryption key for the key server communication as well as the key for decrypting the content will be available in the memory of the device running the

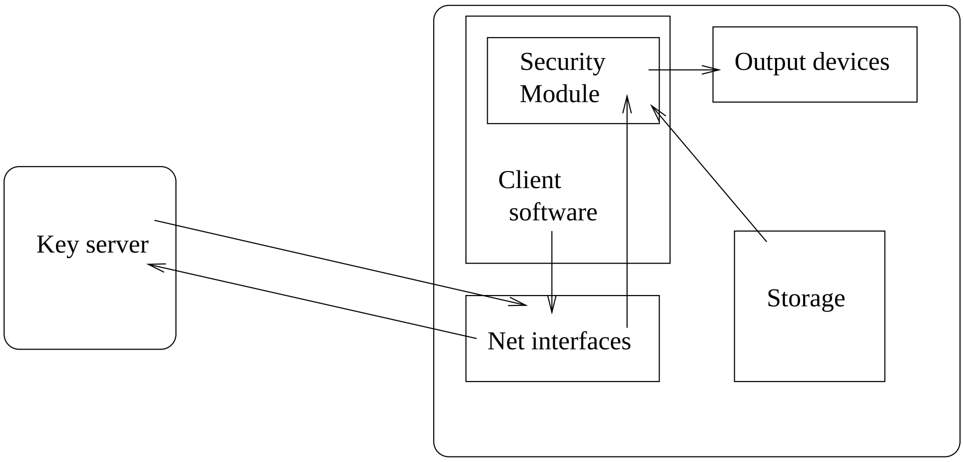

Figure 16: Security module in software

security module. For example, if the security module is run as a process in a UNIX-like operation system such as Linux, the complete memory image of the security module will always be available to the user of such a system. While code obfuscation of a program stored on disk may be somewhat efficient, the obfuscation of a running process may be much more difficult.

Hardware security module

Figure 17 illustrates a security module implemented completely in hardware. The hardware security module performs the same functionality as the software module. However, as the module is implemented completely in hardware, the methods of software reverse engineering cannot be used to extract the keys. The hardware module also allows different levels of security for the keys and data, depending on the security requirements demanded by the content providers and distributors. If high resistance against interception is needed, the security module can be implemented in tamper proof hardware. If less protection is sufficient, the design of the security module can be simplified accordingly.

Security optimizations

If the functionality of the security module is limited to removing encryption from the content, a user can simply store the output of the security module once the content has been decrypted. Additional features may be needed in the security module to make illegal use and illegal distribution less tempting.

Digital watermarking is a technique which can be used to add information such as copyright messages to content in a way that it cannot be easily removed. Optionally, the security module can add a watermark to data being decrypted. The watermark identifies the user whose security module has been used to decrypt the data. If decrypted content is later found in illegal distribution, the content can be traced back to the user who released the unencrypted illegal copy.

Watermarking does not make it more difficult to make copies. However, if removing the watermark reliably from content is difficult the watermark may work as a deterrent against illegal distribution of content.

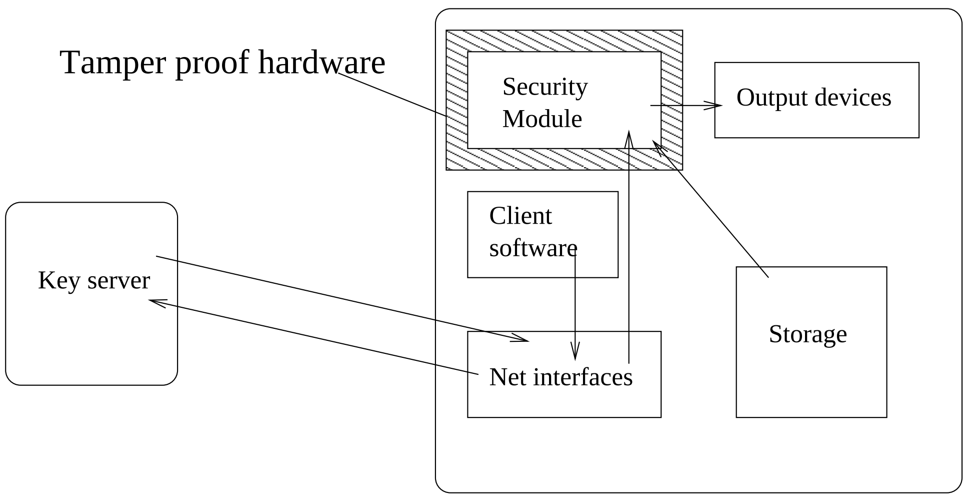

Figure 17: Security module in hardware

Another possible optimization against illegal copying is to make the content more difficult to distribute by making illegal copies significantly larger than the original if the copier wants to preserve the quality of the video. The security module, instead of outputing raw data such as an MPEG stream or an MP3 file, may output content in decoded format. The decoded format for a video clip can consist of video bitmaps ready to be shown on screen. A user who tries to store the video on disk needs to encode the bitmaps again to create an encoded video stream before distributing the copy. The hurdles against re-encoding a video of audio stream are the following.

1. The size of the content increases every time the content is re-encoded if the quality is required to remain even roughly equal to the original.
2. The quality of the content worsens every time the data is encoded again. In some cases it may be possible for the security module to add extra information into the decoded content. The extra information may be designed to be invisible or inaudible to human listeners, but will make the re-encoding of the content more difficult.
3. The security module can insert digital watermarks into the data. The watermark is generated within the security module and added into the content before it is output from the security module. The watermark data identifies the identity key of the security module. The watermark should be impossible or difficult to remove from the content without severely damaging the quality of the content.

The obstacles against the creation of illegal copies are twofold. First, the increase in file size and decrease in quality are technical obstacles against illegal copies. The user planning on using an illegal copy needs to consider whether the decreased quality and increased size of the content are adequate reasons for purchasing a legal license for the content. The second obstacle against creating illegal copies is legislational. The addition of a watermark into the data ensures that the security module used for decrypting the content can be identified from the illegally distributed content.

8.3 Availability

The architecture needs to be able to transmit very large files and streams across public networks. Such a system necessarily has several potential vulnerabilities against Denial of Service attacks. We assume that the protocol has been implemented correctly, and all the potential attacks discussed here are directed only at the protocol itself.

Server-cache traffic

The traffic between the server and the caches is usually sent over a public network. This allows anyone with connectivity to a network to send packets into the same network. The architecture should therefore be able to tolerate any possible attack performed by any member of the network.

All packets delivered from the server to the caches are protected by a signature. Only packets with a signature which can be verified by the caches will be stored into the cache. Therefore, a cache cannot be made to forward packets that are not created by a legitimate server whose key has been configured into the cache. Since a hostile node cannot create new packets that will be acceptable to the multicast caches, the hostile node is left with two alternatives. The hostile node can either replay packets which it has stored earlier, or it can attempt to overwhelm the network or the caches with a large number of packets.

The replay attack with old packets is started simply by capturing a large number of legitimate multicast packets from the network. Later the packets are sent again into the network. Since the packets have originally been created by a legitimate multicast server, the signatures on the packets will be acceptable when the authenticity of the packets are verified by nodes in the network.

However, replayed packets can be easily discarded by multicast caches.

1. Every packet can contain a timestamp which indicated when the packet was created. Packets with too old timestamps are rejected.

2. Caches maintain information about packets that have already been received. If a packet with the same source and destination address and sequence number has already been received, the duplicate is dropped.

These two rules alone enable the multicast caches to drop replayed packets without forwarding or storing them. The second rule alone allows the multicast cache to drop any duplicate packets. However, the first rule can be used to restrict the storage requirements in caches. Rule 2 alone requires a multicast cache to store the identities of received packets indefinitely. Such a database would eventually exceed the storage capacity of any cache. However, rule 1 allows the cache to discard information about packets whose timestamp has expired. Therefore the replay information for those packets can be discarded, and the packets can be dropped based on rule 1 alone. The same functionality can be performed by any router in the network.

Another possible attack between the server and caches is a denial of service attack based on flooding the network with garbage packets. The packets may contain any data as long as the packets consume enough capacity to cause a significant percentage of legitimate packets to be dropped in the network. This type of attack can be most efficiently performed through a distributed denial of service attack, where a large group of compromised nodes simultaneously start sending packets to a certain destination.

The effect of DDoS attacks can be mitigated, for example, through some Quality of Service method which prioritizes authenticatable multicast packets. Details of such QoS protocols are, however, outside the scope of this dissertation.

A flooding based DoS attack can also be directed specifically against the protocol being used. In the case of our architecture, the authentication algorithm being run in the cache nodes is the only part of the protocol which cannot run at wire speed capacity on ordinary hardware. In this attack, the hostile nodes create packets that are designed to consume as much processor capacity from the caches as possible. Even if the hostile nodes are not able to create forged packets which are acceptable to the caches, the reception of legitimate packets can be hindered by the verification of forged packets. Specifically, in this attack, the hostile nodes send packets which have a seemingly legitimate identity. That is, the packets have source and destination addresses which correspond to ongoing legitimate multicast streams. Furthermore the packets need to have a header structure of a legitimate packet of the multicast architecture, and the packets also need to have sequence numbers which have not yet been received by the caches. If the sequence numbers in the packet are chosen in a way that no legitimate packets from the same authentication tree has yet been received, the cache needs to compute the signature for each received hostile packet.

To filter out all the hostile packets without unnecessarily dropping any of the legitimate packets, the cache would need to be able to verify signatures at the rate which packets are received. The problem can thus be solved at least in two alternative ways. First, the multicast cache can have some hardware that is specifically designed to verify signatures at line speed or faster. The second alternative is to handle the filtering higher up in the network. Routers can also be used for filtering packets the same way which is available to the caches.

Cache-terminal traffic

The second leg of the packets path from the server to the terminal is the trip between the cache and the terminal. Most problems that are present at the leg between the server and the cache are also present here. However, as the network capacity at the last hop interface, which may be wireless, is usually much smaller than in the fixed network, the cache terminal interface is usually more sensitive against attacks. On the other hand, the packets sent over the last hop interface are also usually processed by a multicast cache where illegitimate packets are dropped.

Another functionality of the cache is to provide retransmission service to terminals which require one or more packets to be retransmitted. The request protocol and the retransmissions are another target for attacks against availability. An attacker can try to request the multicast cache to make unnecessary retransmissions which can limit the throughput of legitimate retransmissions. Sending forged requests must therefore be made as difficult as

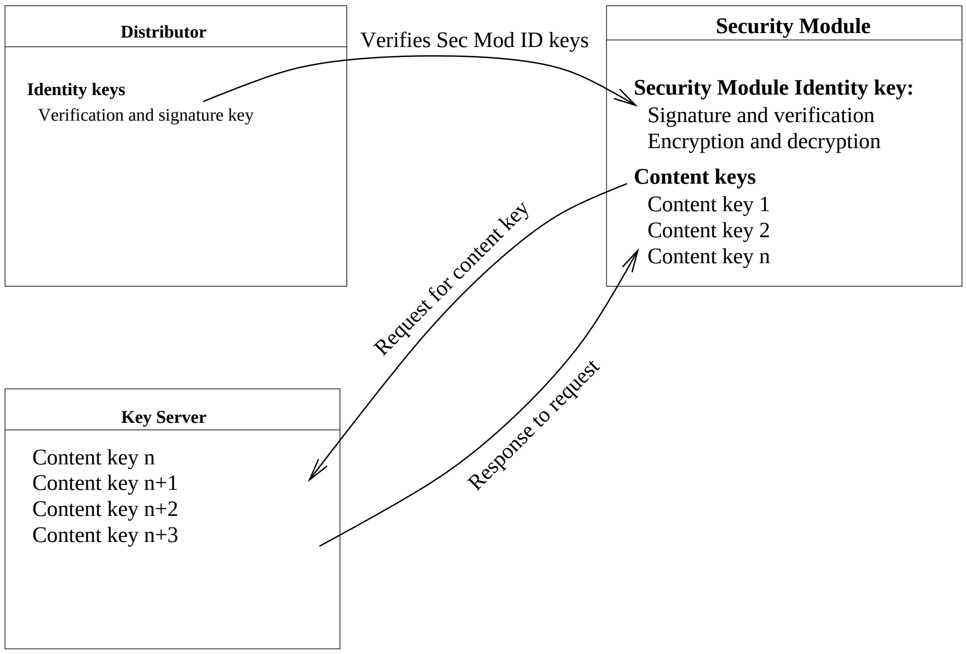

Figure 18: Key management in the architecture.

possible.

Because the multicast caches can also operate over unidirectional links, the caches need to be able to receive requests, not only from the link used for sending retransmissions, from the fixed public network. The requests are sent from terminals which want the retransmissions to be sent over a unidirectional link. To prevent fraudulent requests to be sent from anywhere in the fixed network the multicast cache uses a simple protocol to ensure that the requesting terminal is in an area where the retransmissions can be received.

Each cache announcement message contains a random number. Whenever a new cache announcement message is sent, a new random number is generated for it. A terminal requesting retransmission adds into the request the latest random number received from the cache to which the request is being sent. A cache receiving a request message confirms that the random number in the message equals one of the random numbers recently sent in an announcement. If the verification fails, the request is dropped. Otherwise the request is accepted.

This simple protocols guarantees that the request was generated by a node which is able to listen to the retransmission messages. On the other hand, a hostile node residing anywhere else is unable to send an acceptable request message as it cannot receive the cache announcements which contain the random number. However, a hostile node residing in the area where cache announcements are sent can still generate forged requests.

8.4 Security module failures

Figure 18 illustrates the various keys in the architecture and their purpose. In this section, we discuss the effects of the keys being revealed to an adversary. We also discuss some ways which can be used for mitigating the damage caused by the revelation of the keys.

Content key

A content key is used for encrypting content before it is sent to the network. It protects the content from being used without authorization from the content provider. If a content key is leaked, the content protected with the key can be decrypted by anybody with access to the key.

The content can then be distributed in two different ways.

1. The content can be decrypted, and the decrypted content is placed in distribution, for example, in a P2P network.

2. The decryption key to the content is made available. In this case, a user can download the encrypted content legitimately from the network. However, instead of buying a legal key for the content, the user uses the illegally published key for removing the protection.

Content which is useful for a long time is a more vulnerable target for content key attacks. On one hand, content such as music and movies can maintain its value for years after being published. On the other hand, the value of news quickly diminishes after publication. Additionally, the market value of a movie is much larger than the value of one piece of news. If the cost of downloading content legally is sufficiently small, the reward gained from using content without authorization diminishes.

To diminish the problem, each content should have a separate content key. If one of the content keys is revealed, only the one content can be decrypted with the key. However, if a content key is revealed, the content provider can respond by switching to another content key for the content. In this case, the content provider sends a new copy of the content to the network. Any packets already available in caches are replaced with the new packets.

When the content key is changed during the lifetime of a content, users can end up in a situation where the content they have downloaded consists of packets encrypted with several keys. When the user requests the decryption key, the key server returns all of the keys that have been used for encrypting the content. A user with authorization can still use the packets encrypted with the old key. However, the revealed old key is no longer sufficient for decrypting the content if some of the received packets are encrypted with the new key.

If an unauthorized user is able to receive enough packets that are encrypted with the revealed key, the content can be used illegally. However, if some of the packets are encrypted with the new key, the user must request the new key from the key server.

Content provider key

The content provider key is used for signing every content packet that is sent over the network. The signature ensures that the receiver can verify the creator of the content even when some or all of the packets have been received from an untrusted middleman.

An attacker who has gained access to a content provider key can create content which appears to originate from the content provider whose key was revealed. The attacker can create false content, modify existing content or send old content as new content.

A method for revoking content provider key is needed. If a content provider key is revealed, it is added to a revocation list which is maintained by a trusted third party (TTP). The receivers periodically download the list from the TTP. If a content is signed by a key which is included in the list, the device indicates that the content can no longer be trusted. The TTP can also be used for retrieving verification keys for the content provider keys. If an old content provider key is revealed, a new verification key for the content provider's new content provider key can be distributed by the TTP.

Identity key

An identity key is located in the security module in every receiver. The identity key is used by the key servers for identifying receivers that are requesting content keys. If the private part of an identity key can be extracted from the security module, it can be used for impersonating another receiver's security module. The key can then be used for the following attacks.

1. The key can be used for downloading and using content as the user whose identity key was extracted.

2. The extracted key is used for breaking the communication between the security module and the key server. The content key for the content can then be extracted from the communication.

The first attack is fairly straightforward to detect. The content provider needs to maintain statistics on the behavior of identity keys. For example, if an identity key is used simultaneously from several places, which are physically far away from each other, the key can be placed under surveillance. If the unusual behavior continues, the key can be revoked.

The second attack is more difficult to detect and handle. Every receiver uses the same content key for decrypting the content. If a content key is extracted from the communication, the content key is the same regardless of which identity key is used to decrypt it. The content provider cannot identify the identity key which was used for extracting the content key. However, the content key can still be used for decrypting the content by any receiver.

The most important protection against identity key attacks is to create identity modules which are resistant against key extractions. If possible, the security module should be implemented as a tamper proof hardware.

Leak tracking

Content can never be entirely impossible to copy and distribute illegally. Even if other methods are effectively prevented a movie can still be recorded with a video camera from the display in the receiver. Similarly, music can be recorded by microphone directly from the speaker if the receiver device does not offer a line out.

Content copied in this way will usually suffer some amount in degradation in quality compared with the original content. However, the quality may still be sufficient that some users are willing to save even a small amount of money by using an illegal copy.

Such attacks can be mitigated by including a watermark into the content when the security module decrypts the content. Several watermarking algorithms exists for several types of content. Further discussion regarding watermarking is out of scope for this book.

9 SIMULATIONS

9.1 Simulator

In the previous chapters, we presented an architecture based on creating blocks on packets from content that needs to be delivered to a potentially large group of receivers. The architecture describes how the packets are created, encrypted, authenticated and how the FEC data is created in the server. The architecture also described how the data is reconstructed from the combination of data and FEC packets that have been received by the receivers.

For now the architecture has not described in detail the protocol used between the multicast cache and the receivers. There are several possible ways of implementing the protocol. The alternatives include the following possibilities:

1. Unicast. The receiver requests the needed packets over a unicast connection. In this case, each receiver needs a separate connection, and every packet that is sent benefits only one receiver. Unicast can still be a good alternative if only one receiver under the access point needs packets from a block which has already been received by all other nodes. Unicast can be implemented over either TCP or UDP.

2. Multicast. Multicast is a good alternative when content needs to be transferred simultaneously to one or more receivers under the same access point. With multicast, a packet that is sent only once can be received by all receivers within the range of the access point.

3. Peer-to-Peer (P2P). Content can also be delivered directly form one receiver to another. If one receiver already has received the content required by another node, the content can be directly transferred between the two nodes.

We have chosen to simulate only the last hop environment where data is transferred using multicast. We have used a discrete event simulator for the task.

9.2 Simulated environment

Figure 19 illustrates our simulated environment. The simulation consists of the protocol used for transferring the content from a multicast cache to a group of receivers over multicast.

Content delivery

The content delivery over multicast from the server to the receivers can be divided into two separate legs:

1. Delivery from the server to the multicast cache.

2. Delivery from the multicast cache to the receiver.

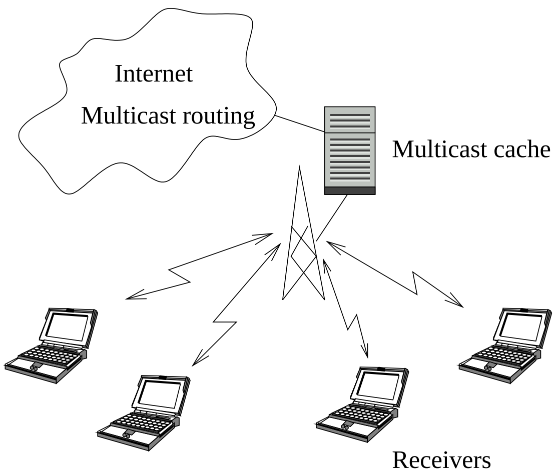

Figure 19: Simulated environment.

The first leg in not explicitly implemented in our simulations. Our simulations assume that the multicast server creates enough packets that the multicast cache will be able to reconstruct the block. As long as the multicast cache receives at least the minimum number of packets for each block, it can recreate the packets that were dropped during the transmission in the first leg. We also assume that the packet delivery between the server and the multicast cache has sufficient capacity and the cost of delivering the packets is very low.

Our simulations consist of the protocol used between the multicast cache and the group of receivers. The goal of the simulation is to simulate the efficiency of our protocol used for the last hop delivery. The protocol should be able to deliver the content to as many receivers as possible using the minimum amount of resources.

Last hop network technology

The simulation assumes a wireless link between the multicast cache and the group of receivers. The last-hop link technology is assumed to have the following properties in the downlink direction.

- **Network technology.** The network technology is able to send multicast packets at the link layer. That is, a packet that is sent by the multicast cache can be received and used by several receivers. If a receiver is not interested in receiving the packet, the packet can be discarded by the receiver's hardware.
- **Network capacity.** The network technology is capable of transferring packets at a sufficient rate to deliver the content.
- **Reception cost.** Receiving multicast packets is inexpensive in the technology used. For example, no RTS/CTS type handshaking for multicast packets is used.

The network technology used for sending packets from the receivers to the multicast cache (uplink direction) is assumed to have the following minimal properties:

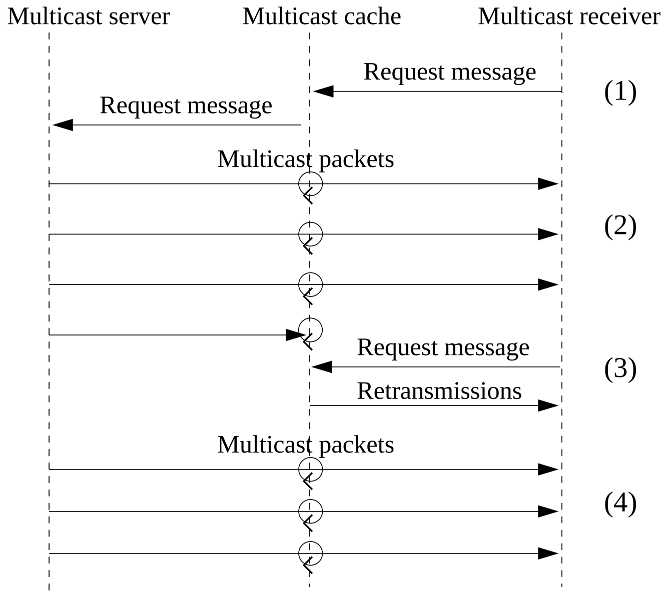

Figure 20: Packet flow with the first receiver.

- **Network technology.** Unicast transmissions are required.
- **Network capacity.** Transmissions are infrequent, and the network capacity needed for them is small compared to the downlink direction.
- **Transmission cost.** Transmission of packets does not need to be inexpensive in terms of energy.

The above lists represent the minimal requirements for the link technologies. If protocols with more suitable properties are available for use, they can be used instead. It is also possible that both the uplink and the downlink delivery is handled by the same protocol as long as all the minimal properties are fulfilled by the protocol in both directions. For example, 802.11a, 802.11b and 802.11g are protocols with such suitable properties.

The multicast cache is assumed to receive its packets from a multicast server located somewhere within the Internet. The operation of the server is not explicitly simulated. The multicast cache is assumed to receive a sufficient number of packets which it can then deliver to the receivers.

Simulation applicability

The packets can be delivered to the receivers in two separate ways. The packets can come directly from the multicast server, or the packets can come from the local multicast cache. In both cases the retransmissions are handled by the local multicast cache. We briefly compare the two cases and discuss their differences from the point of view of the simulation.

Figure 20 illustrates the case where the packets come directly from the multicast server. Four separate situations in the packet flow are identified below.

1. The receiver requests the multicast stream from the local multicast cache. If the multicast cache is already receiving the requested multicast stream, the cache can immediately start transmitting packets to

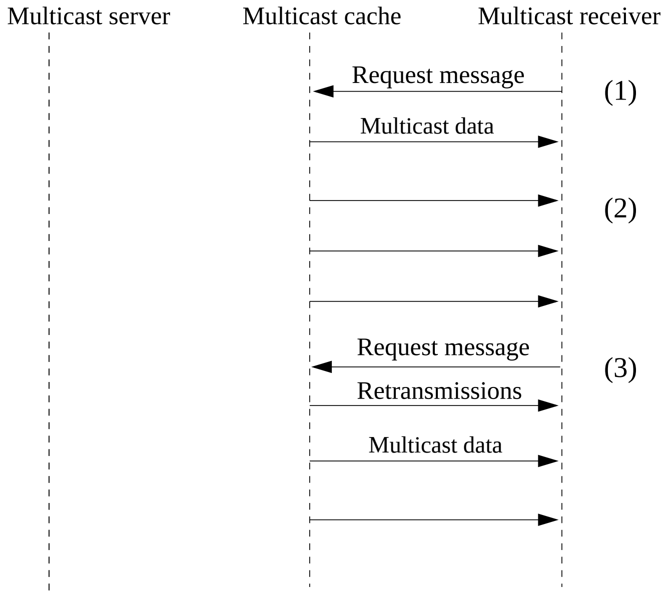

Figure 21: Packet flow with later receivers.

the receiver. If the multicast cache is not already receiving the needed stream, the cache sends a request to either the multicast server or another multicast cache. In the example, the request is sent directly to the multicast server.

2. The multicast packets are received directly from the multicast server. The multicast cache also receives copies of the packets and stores them on disk.

3. If retransmissions are required, the receiver requests the needed packets from the nearest multicast cache. The original multicast server does not know about the retransmissions.

4. More packets are received directly from the multicast server.

The retransmissions in step 3 are sent in addition to the packets coming directly from the multicast server. That is, the flow of packets from the server is not interrupted, but the retransmissions are sent in addition to the packets coming from the server.

Figure 21 illustrates the packet flow in a situation when the multicast cache has already received the packets that are requested by the receiver.

1. The receiver requests the multicast stream from the multicast cache.

2. Since the multicast cache already has received the packets that are requested, the multicast cache is able to satisfy the request. The multicast server does not participate in sending the packets.

3. Any necessary tretransmissions are handled locally between the multicast cache and the receiver.

In the first example, the packets are received directly from the multicast server, whereas in the second example the packets are received from the multicast cache. In both cases the packet flow seen by the receiver is the same.

Although, the simulations are designed for the case where the packets are sent by the multicast cache, the results are also applicable to the case when packets are received directly from the multicast server. The packet flow is similar in both cases. The only difference is the small increase in delay if the packets need to be requested from the server instead of the cache.

9.3 Simulation input

The following are the most important input parameters that are configured into the simulator before starting a simulation run.

1. **Number of receivers.** The number of nodes which receive content directly from this sender.

2. **Receiver drop profile.** The receiver drop profile configures the probabilities of packets being dropped before they reach the receiver. The probabilities can be configured independently for groups of receivers, and the probabilities can be set to change over time.

3. **Data packets in block.** The number of data packets from which the FEC packets are computed.

4. **Default number of FEC packets in block.** The number of FEC packets sent per block by default. If the `autofec` parameter is set, this parameter is used as the starting value for the autofec algorithm.

5. **Autofec.** Set to either on or off. If autofec is on, the sender algorithm attempts to adjust the FEC to the optimal value for the current group of receivers. If autofec is off, the default number of FEC packets is always used.

6. **Fast request.** Set to either on or off. If set to on, receivers can send retransmission requests before the current block is completed by the sender.

7. **Maximum FEC.** The maximum number of FEC packets that can be sent for one block.

8. **Number of blocks to send.** The number of blocks to transfer in the simulation.

9.4 Protocol description

The architecture was presented from a high level point of view in Chapter 6. Everything that was described about the architecture in that chapter still applies. In that chapter the architecture was described in a very generic way without committing to any unnecessary details. Here, we present the details of our architecture that have been adapted for use over a multicast capable last hop connection.

Table 1: Variables used by sender algorithms.

`blocksize`	Number of sequence numbers reserved for one block.
`curblock`	Current block number being sent.
`intraseq`	Intra block sequence number. That is, the sequence number of the packet within the current block.
`fecs`	Number of FEC packets currently sent for one block.
`limiter`	Current limiter value.

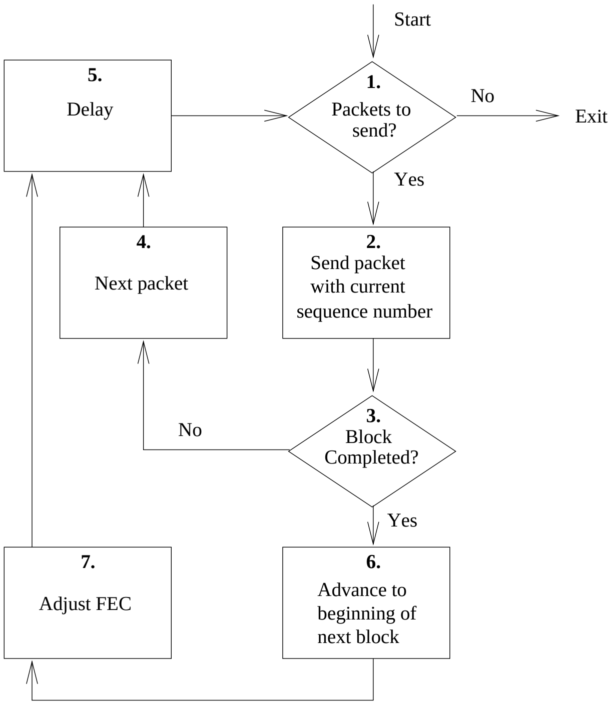

Figure 22: Algorithm for the sender's packet sender thread.

We present the algorithms as flowcharts. The operation of the multicast cache and the receiver are presented separately. Each flowchart is accompanied by an explanation which describes the flowchart in more detail.

9.5 Sender

Table 1 illustrates the variables used by the sender algorithm. The `limiter` variable is regularly sent by the sender to its group of receivers. The variable controls the rate at which request messages can be sent by the receivers. Large values for the variable discourage receivers from sending requests.

Packet sender thread

The packet sender thread sends the multicast packets from the sender. Figure 22 illustrates the algorithm used for sending packets from the sender to the receivers. Before the thread is started, a few variables need to be initialized to their initial values. The initialized variables and their values are: `curblock` = 0, `intraseq` = 0, and `fecs` = 0, `blocksize` = constant.

1. Determines whether or not it has more packets to send. If all packets have been sent, the sender algorithm exits.

2. Sends the next packet in stream.

3. Test whether the packet sent in state 2 was the last packet in this block. The packet was the last packet in the block if all data packets and all FEC packets from the block have been sent.

4. Advance to the next packet within this block. `intraseq = intraseq + 1`. If the packet is not available, send the next available packet from this block.

5. Delay to limit the packet transmission rate. `sleep(intrapacket delay)`.

6. Advance to next block. Adjust the sender variables as follows. `intraseq = 0; curblock = curblock + 1;`

7. Give new value to the `fecs` variable. If fast requests were received during this block, the variable is set to the value computed in the request receiver thread. If no fast requests were received, the value is decreased by the number of continuous blocks during which no fast requests have been received.

Request receiver thread

Figure 23 illustrates the sender's retransmission request receiver thread. The thread processes requests received from the receivers, and decides how the sender should respond to the received request.

1. Wait until either a request packet or a fast request packet is received.

2. Is the received packet a fast request packet? If `YES` goto 5, else goto 3.

3. Add requested packets to the end of the retransmission queue.

4. `limiter = limiter * 1.03;`.

5. Is the received fast request for the block currently being sent? That is, is `curblock` $\leq$ `seqnum` $<$ `(curblock + 1)` $*$ `blocksize`? If `YES` goto 6, else goto 1. That is, if a fast request is received for a block which has already been completely sent, the fast request is discarded.

6. If the number of FEC packets requested by the request is larger than the current value for FEC packets, increase the `fecs` variable to the new value. If the new value for `fecs` is larger than the number of FEC packets available, the value is limited to the value of available packets. That is, `fecs = min(max(requestedFecs, fecs), maxfecs)`.

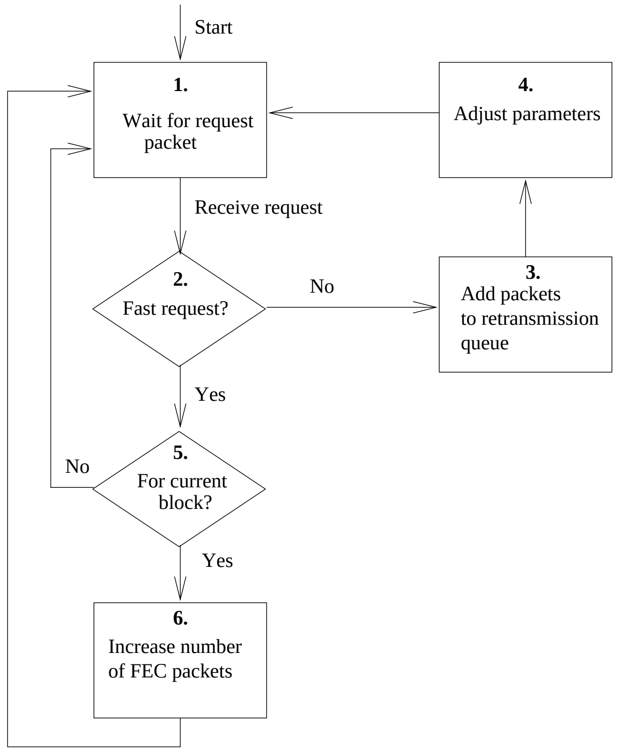

Figure 23: Sender's retransmission request receiver thread.

Retransmission sender

Figure 24 illustrates the sender's retransmission sender thread. The responsibility of the thread is to retransmit queued retransmission packets at a constant rate.

1. Pop the first sequence number from the start of the retransmission request list.
2. Select the packet to be sent. The packet to sent is any one of the unsent FEC packets from the block of the sequence number. If all FEC packets have already been sent from the block, send the packet whose sequence number was received in the previous step.
3. Send the packet.
4. Intrapacket delay for retransmissions.

Limiter adjuster

The goal of the limiter is to limit the number of request messages sent by the receivers. The sender regularly tranmits the limiter value to the receivers. If the sender receives too many request messages, the value for the limiter is increased. An increase in the limiter value signals the receivers to send fewer requests. Once the number of request messages from the receivers is small enough, the value for the limiter can be decreased.

Figure 25 shows the sender's limiter adjustment thread. The goal of the thread is to decrease the value of the limiter. The limiter is increased in the

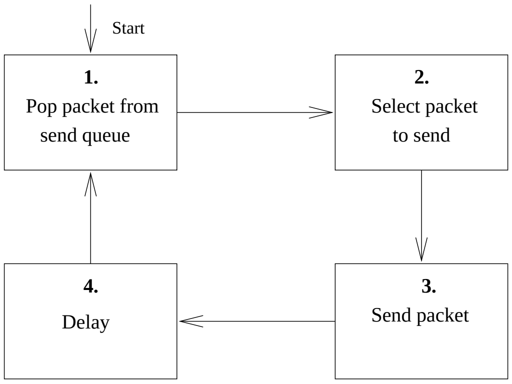

Figure 24: Sender's retransmission sender thread.

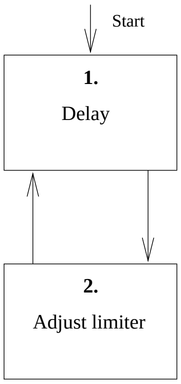

Figure 25: Sender's limiter adjustment thread.

request receiver thread every time a request packet is received. Together, these two threads maintain a suitable limiter value.

1. Delay (1s)
2. `limiter = max(1, limiter * 0.99)`

9.6 Receiver

The receiver threads are run on the devices that wish to receive content from the multicast caches. If the device does not automatically receive suitable packets from the sender, the receiver threads are responsible for requesting retransmissions for the missing packets. The receiver threads also send feedback to multicast caches in the form of request messages. The caches use the feedback to estimate the number of FEC packets per block that are required by the receivers.

Receiver thread

Figure 26 shows the receiver's packet receiver thread. The responsibility of the receiver thread is to store received packets. This thread also sends fast request packets when it detects that an insufficient number of packets for the current block is expected. The receiver thread algorithm is as follows:

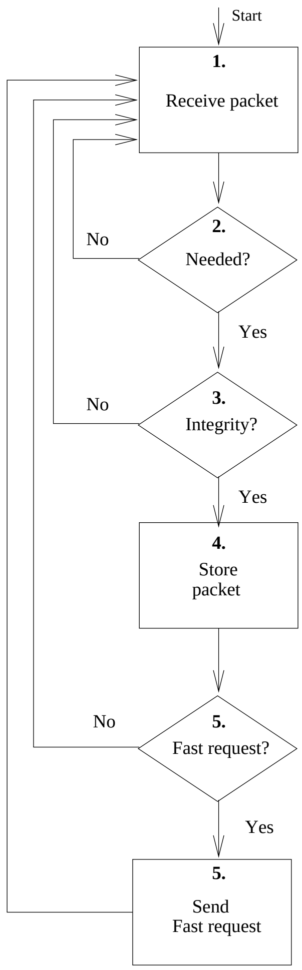

Figure 26: Receiver's packet receiver thread.

1. Wait until a content packet is received.
2. If the packet is needed, goto 3, else go back to 1. A packet is needed if it has not yet been received by this receiver and the packet is for a block which has not yet received a sufficient number of packets to be reconstructed.
3. Verify integrity. The receiver verifies the signature in the packet. If the signature is valid, goto 4, else goto 1.
4. Store the received packet.
5. Is a fast request needed? A fast request is needed if the number of packets expected from the sender is less than the number of packets needed to complete the block. If `YES` goto 5, else goto 1.
6. Send fast request packet. The sequence number in the packet is set to the value of the first missing packet. The number of packets in set to the total number of FEC packets the receiver wishes the sender to send.

Request sender
The receiver's request sender in Figure 27 sends retransmission requests for blocks that were not completed when the block was originally sent by the multicast cache. The receiver's request sender limits the number of requests based on the limiter value received from the multicast cache. The request sender algorithm is as follows:

1. Delay. This delay limits the rate at which the loop consisting of the steps 1, 2 and 3 can be executed. If no packets are needed in step 3, the algorithm waits a short time to avoid running in a busy loop through steps 1, 2, and 3.
2. Compute the set of packets that are needed to complete the blocks which have been completely sent by the sender but which are still missing in this receiver. The set of missing packets is denoted by N_1.
3. Is the set N_1 nonempty? That is, are any packets needed? If `YES` goto 4, else goto 1.
4. Delay. The length of the delay is determined by $T = \frac{R*L}{\sqrt{D}}$, where T is the delay in seconds, L is the current limiter value, and R is a random number between 0 and 1. D is the difference of the largest received sequence number and the smallest sequence number which is missing and needed.
5. Compute the set of packets needed to complete block which have been completely sent by the sender but are still missing packets. The set of packets is demoted by N_2.
6. Compare the sets N_1 and N_2. If they are equal goto 7, else goto 1.
7. Send request message with requests for the packets in the set N_1.

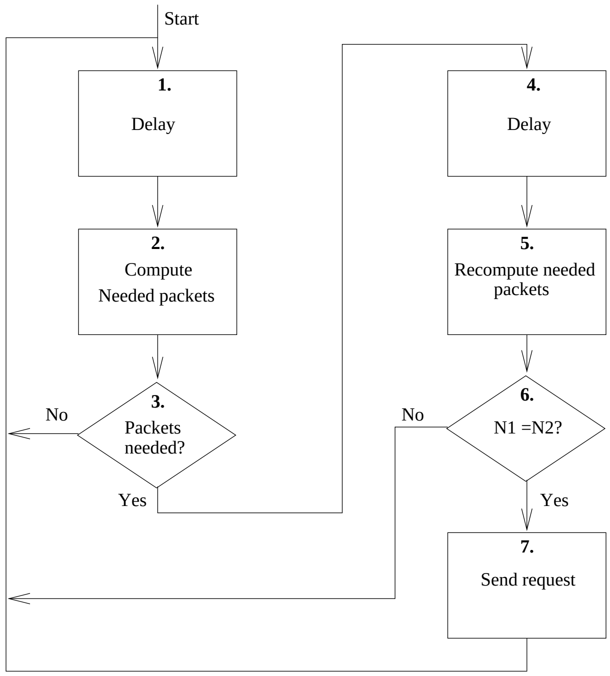

Figure 27: Receiver's request sender

The goal in the algorithm is to send as few request messages as possible while maintaining as small a delay as possible in receiving all the needed packets. In step 3 in the above algorithm, the receiver computes a set of packets it needs from the previously sent blocks. The same computation is performed in every receiver in the group of receivers. However, the request that was computed in step 3 should not be used for sending a request immediately. If the group of receivers is large, a large fraction of the group may require packets from the same block. If every receiver is allowed to immediately send a retransmission request, the sender can be flooded with overlapping requests. Instead, every receiver delays sending the request for an amount of time that depends on:

- The rate of requests being processed by the server. A large value for the limiter L, indicates that the server is processing requests at a high rate. Therefore, some other receiver is likely to request the packets that are needed also by this receiver.

- The urgency of the missing packets to the receiver. If the packets are a large sequence number distance from the highest transferred sequence number, the delay is decreased.

- Random wait. A randomization factor limits the number of identical requests being sent immediately.

After the delay, the set of needed packets. $N2$, is recomputed. If the new set of packets differs from the set $N1$, the receiver assumes that at least some of the needed packets have been retransmitted in response to a request from another node. In this case the request is not yet sent. Instead, the receiver wait for more packets to come.

9.7 Measured variables

Transmission cost and delay are the two main variables used to measure the efficiency of the system. The variables were chosen because they represent properties of the system that are directly visible and important to the users of the system. The definitions of transfer cost and delay are as follows.

Transfer cost. The transfer cost represents the resources used by the system to deliver the content over the last hop link. The cost consists of the radio resources both in the downlink and in the uplink direction. The relative cost of transmission in the uplink and downlink direction may be different.

Delay. The delay measures the time between when the content was given to the system and the time it is received as usable data at the receiver. The delay depends on several parameters that also affect the transfer costs of the system.

Additional helper variables are also used. These variables are not directly important to the users of the system. However, they are useful in understanding factors that affect the delay and transfer cost variables.

Block size. The block size determines how many packets of content are used to compute each FEC block. Increasing block size increases delay because more data needs to be buffered before data can be sent forward. However, increasing block size can decrease transfer costs by giving improved protection against packet loss.

Limiter value. The limiter limits the rate at which receivers send requests. The goal is to reduce the number of unnecessary requests when a large number of receivers are receiving content from one sender. Limiting the number of requests decreases data sent in the uplink direction at the link between the receiver and the access point. However, the limiting mechanism can also increase delay.

9.8 Simulation default settings

Table 2 lists the default values for the parameters used in our simulations. These values are used in every simulation run unless we have specifically mentioned otherwise.

9.9 Transfer cost

The transfer cost estimates the network capacity cost needed to transfer content to the group of receivers.

The transfer cost is based on the number of packets sent in order to transfer the content to all of the receivers. The cost of sending one packet in the downstream direction is assigned with the relative cost 1. Sending one packet in the upstream direction is assigned the relative cost 5. The total cost is calculated as the sum of the costs of transferring the packets in the downlink and in the uplink direction.

Table 2: Default parameter values for simulations.

Number of blocks to send	100
Data packets in one block	100
Total packets sent	10000
Packet payload length	1000 bytes
Packet rate	5 packets/second
Retransmission packet rate	10 packets/second
Error percent	15 (random loss, constant)
Packet payload length	1000 bytes
Autofec	ON
Fast request	ON
Default FEC	10 packets / block
Receivers	100

Number of receivers

Figures 28, 29 and 30 illustrate the transfer cost as the function of the number of receivers. In Figure 28, the simulations were run with a random packet error rate of 10%.

When the number of receivers is very small, a lower limit for the number of packets transmitted by the multicast cache can be computed. The theoretical limit to the number of packets can be computed as $\frac{10000}{0.9} \approx 11100$ packets, assuming content consisting of 10000 packets with an error rate of 10%.

This estimate does not include the cost of sending the multicast cache feedback packets which utilizes network capacity in the uplink direction. Neither does the estimate include the cost of the additional FEC packets which are needed to protect against variations in the number of packets that are actually lost in each block. Still, the mathematical estimate and the simulated results give compatible results.

The total cost increases fairly rapidly as the number of receivers increases from 1 to 200. After the number of receivers increases over 200 receivers, the curve stabilizes to a very gradual increase. Even when the number of receivers is as high as 1000, the total cost of transfer is no more than roughly 50% more than when sending the data to only one receivers.

Figures 29 and 30 illustrate the same simulation with error rates of 20% and 30%, respectively. Both cases show behavior similar to the earlier simulation run, and the curves stabilize to a very gradual increase as the number of receivers increases. The error rate makes essentially no difference in the number of request packets that are sent. The additional cost of retransmissions is almost identical in all of the three figures.

In Figure 31, the cost of transfer per receiver is shown as the function of the number of receivers when the error rate is set at 10%. The figure also shows the total number of request messages sent by all receivers. The cost per receiver drops rapidly as the number of receivers grows.

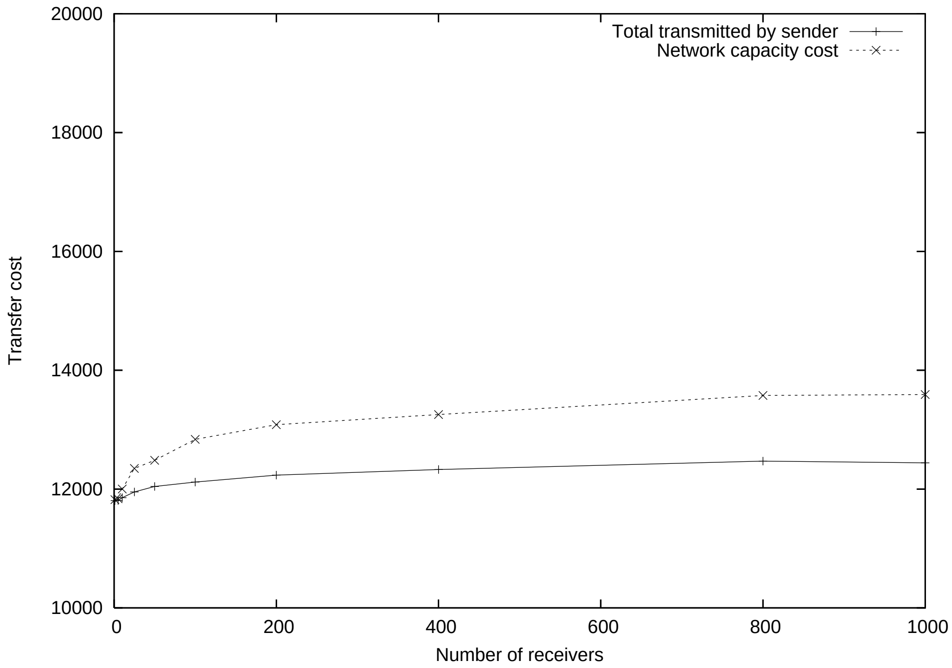

Figure 28: Transfer cost as the function of the number of receivers with packet error rate 10%.

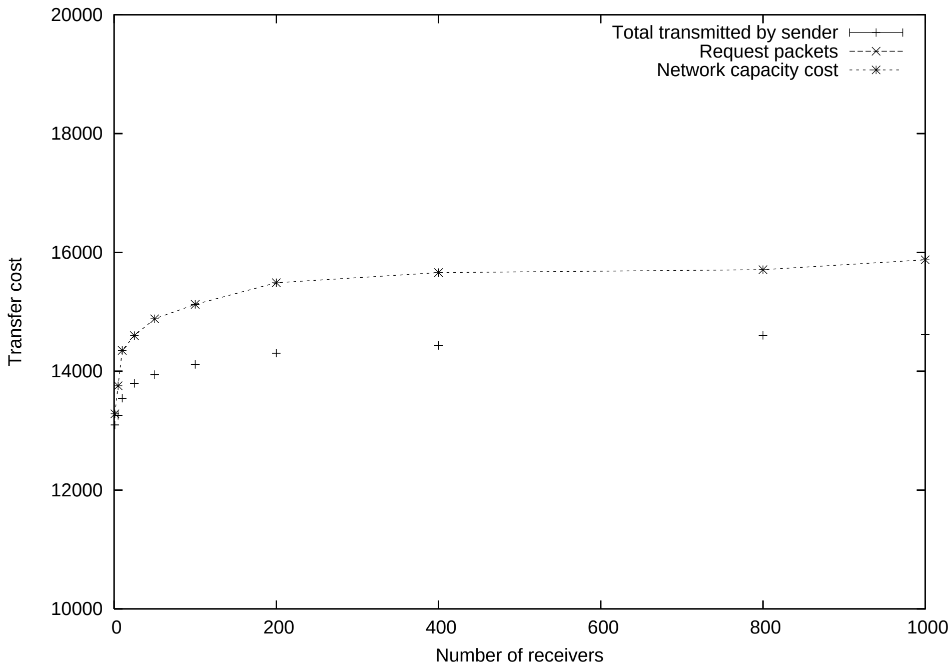

Figure 29: Transfer cost as the function of the number of receivers with packet error rate 20%.

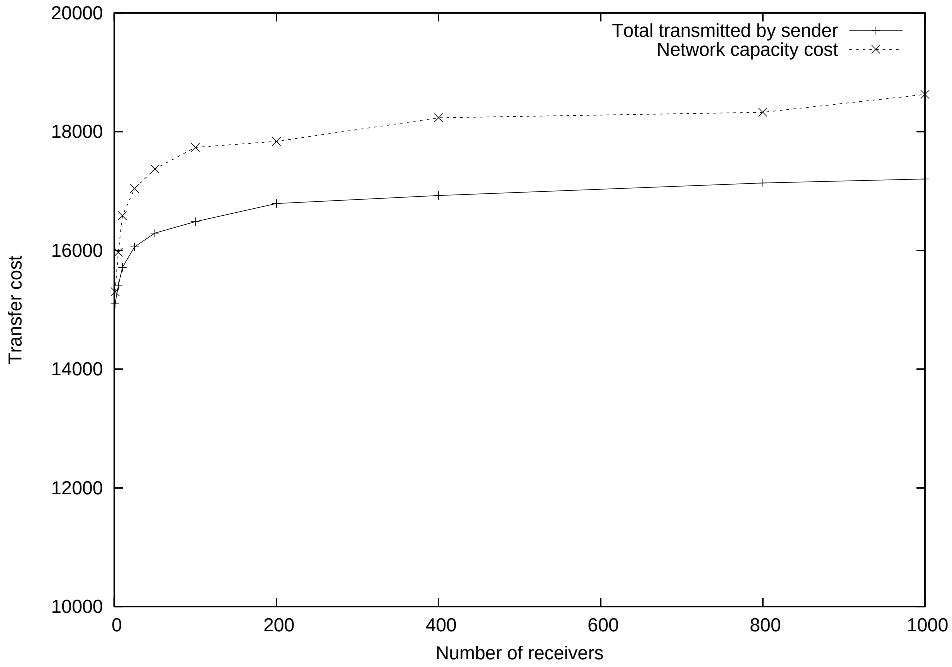

Figure 30: Transfer cost as the function of the number of receivers with packet error rate 30%.

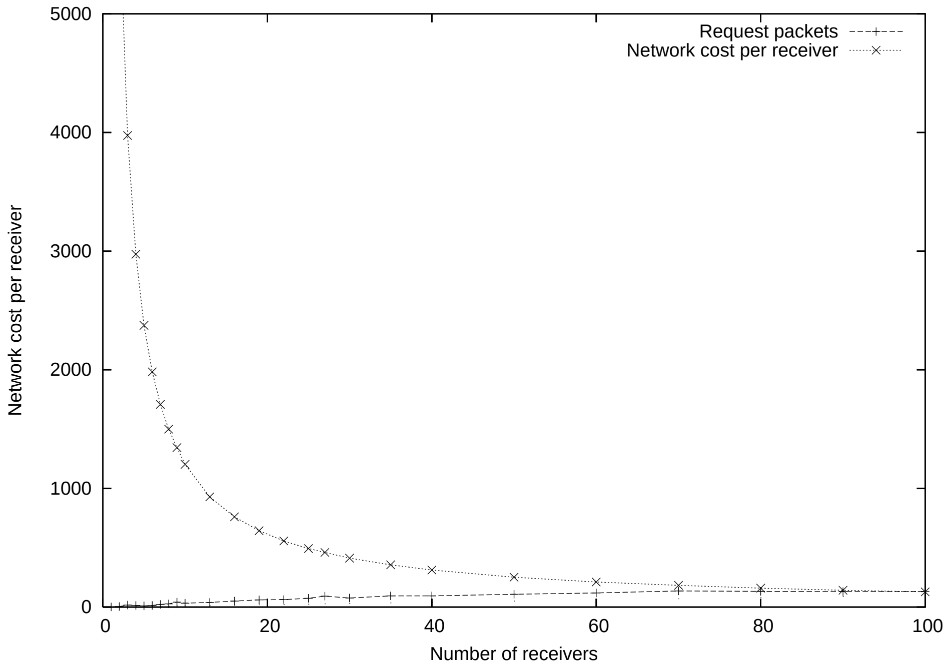

Figure 31: Transfer cost per receiver as the function of the number of receivers. Error rate 10%.

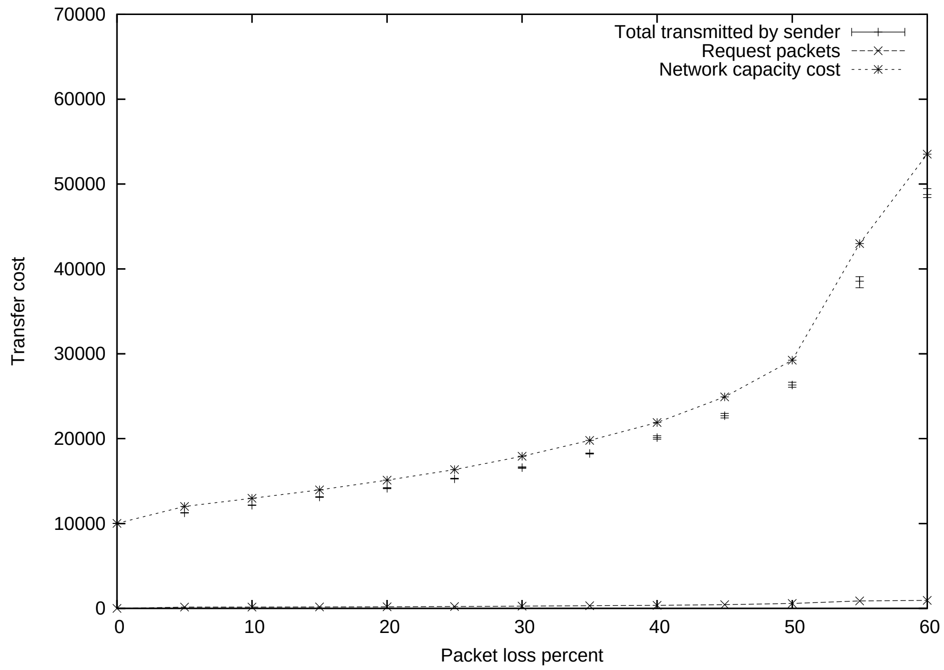

Figure 32: Transfer costs as the function of packet loss percent.

Packet loss

Figure 32 illustrates the transfer costs for 100 receivers as the function of the error percent. When the error rate is 0, the algorithm sends only 10000 packets, which is the theoretical minimum. Is this case, no request packets are sent by the receivers since every receiver always receives all of the packets it needs to reconstruct the content.

The simulations assume content which requires reception of 5 packets/second on average. This rate is assumed to be enough for the receiver to utilize the content as it is received. For example, if the content is an audio stream, the receiver can play the audio in real time. The maximum capacity allowed for retransmissions is 10 packets/second.

The simulation in Figure 32 shows a curve titled Total transmitted by sender. This curve gives the number of packets that were transmitted by the multicast cache during the entire simulation. Since the simulation consists of delivering a total of 10000 content packets to receivers over a 5 packets/second rate during a simulated run of 2000 seconds. Additionally, the multicast cache can send up to 10 packets/second of FEC packets and retransmissions. That is, when the total number of packets sent by the multicast cache during the simulation exceeds 30000 packets, the multicast cache can no longer send the content to all receivers in real time.

In the above simulation, the 30000 packet limit is exceeded when the error rate in the simulation increases above 50%. In this case the multicast cache can no longer send FEC packets and retransmissions fast enough to satisfy the receivers in real time. Retransmission requests start getting backlogged, and some receivers will not receive a sufficient number of packets for every block.

If the error rate increases over the limit only temporarily, the multicast cache is able to deliver the missing content packets to the receivers with an

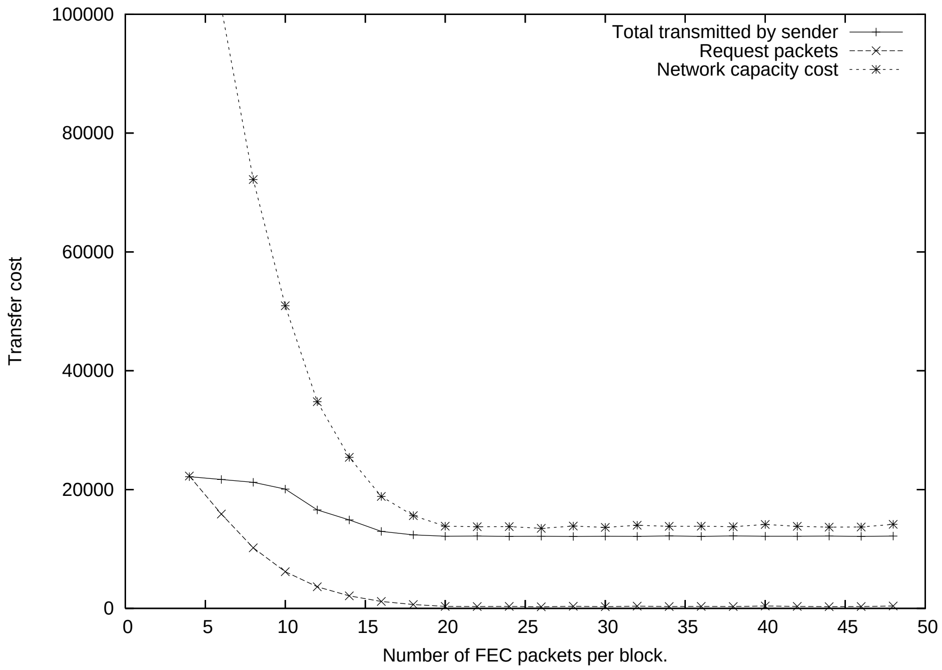

Figure 33: Effect of maximum number of FEC packets to transfer costs.

increased delay. However, if the error rate persists over the limit, the content transfer will fail.

Limited FEC

All of the previous simulations have been run assuming that the multicast cache always has available more different FEC packets than are ever needed. The assumption may not be true if, for example, the content provider attempts to minimize the number of packets sent by the multicast server.

Figure 33 shows the cost of transferring content to receivers as the function of the number of FEC packets available for each block. The number of receivers is set to 100, and the error rate for the transmission is 10%. When the number of FEC packets per block is less than 15, the cost of additional retransmissions is significant because:

1. The multicast cache cannot increase the number of FEC packets to a sufficient level. Therefore, the receivers will send fast requests for every block.

2. Retransmissions cannot be performed with unused packets, as all of the packets from the block have already been sent at least once. If all of the FEC packets are exhausted even before retransmissions are sent, the multicast cache must start resending packets it has already sent once. This results in a situation where receivers do not benefit from packets that are sent in response to requests from other receivers.

The number of FEC packets available to the multicast cache should be large enough that the multicast cache rarely runs out of FEC packets. This can be accomplished by creating and sending enough FEC packets in the multicast server. The number of packets needs to be large enough that a sufficient number of packets for each block is received by the multicast cache even after packet loss in the network between the server and the multicast cache is significant.

The number of FEC packets generated in the multicast server should be large enough to ensure that multicast caches do not run out of FEC packets for any block. If the number of FEC packets is too low, the data will be delivered to the receivers, but the cost in network resources increases significantly.

Block size

Block size is the number of consecutive packets from which the FEC data is computed. The default value for block size in the simulations has been 100 packets. There can, however, be reasons for using either a smaller or a larger value for block size.

A smaller block size reduces delay. The algorithms result in delays in both the server and in the receiver. The delay in the server is caused by the FEC computation which cannot start until all of the packets over which the FEC in computed have been created. In the other end, the receiver cannot reconstruct a block until it has received enough packets from the block to compensate for the packet loss in the block.

On the other hand, a larger block size increases tolerance against errors. A small block size is beneficial when transferring content which can be used while it is being received. Content which is usable only once received in its entirety, can consist of much larger blocks. Examples of such content include, for example, software updates and pictures.

Figure 34 illustrates the cost of transferring a file consisting of 50000 packets to 100 receivers as the function of block size. The error rate in the simulation is set at 10%. Figure 35 shows the same simulation with an error rate of 20%. A block size of less than 100 packets significantly increases the cost of transfer in both figures.

A small block size significantly increases the cost of transfer. A large block size ensures that the number of packets lost for each block remains fairly constant as long as the error rate remains constant. However, when a small block size is used, the number of packets dropped by each receiver in a block varies greatly. Some receivers can receive virtually all packets that were sent in a block while some other nodes lose a very high number of packets. Since the multicast cache always needs to adapt to the behavior of the receivers with the worst loss rate, a very high number of FEC packets must be sent continuously.

Error domination

In order to transfer the content reliably to all receivers, the multicast cache must adapt to the receivers with the worst connection in its area. Even if 90% of receivers in an area are able to receive packets without any packet loss, the remaining 10% of receivers will force the multicast cache to adapt to their optimal settings.

Figure 36 illustrates this case. The figure shows the total transfer cost of 10000 packets to 100 receivers. The 100 receivers are divided into two groups. The group A receives packets with an error rate of 20%, and the group B has an error rate of 10%. The X-axis in the figure gives the total

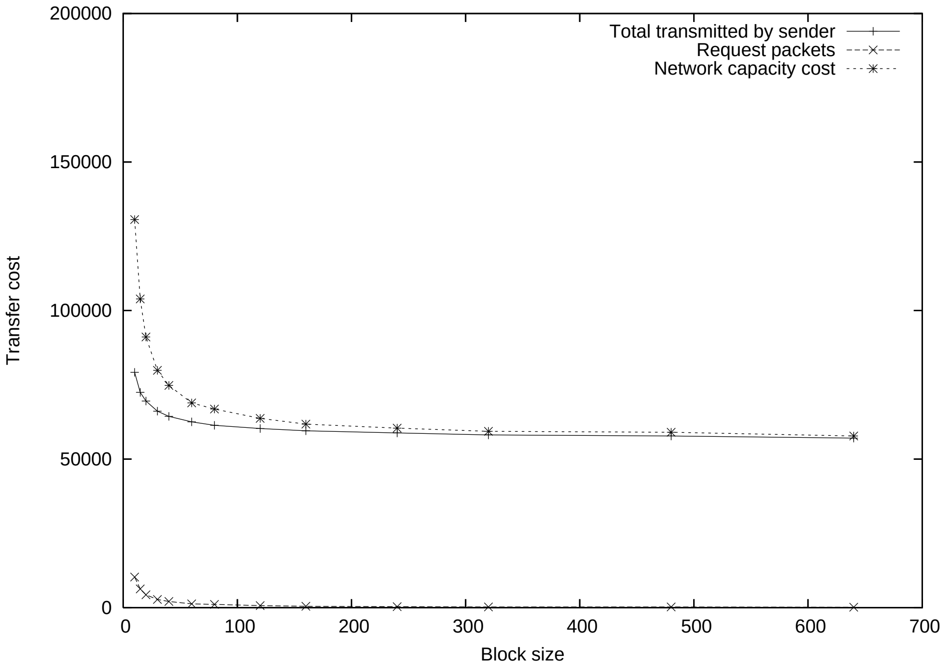

Figure 34: Transfer costs as function of block size.

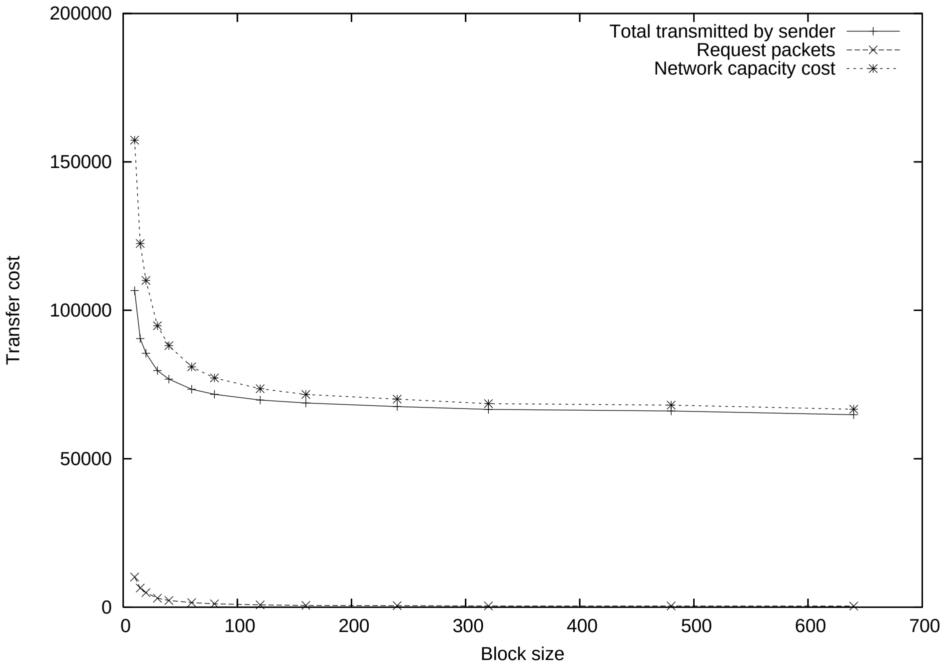

Figure 35: Transfer costs as function of block size.

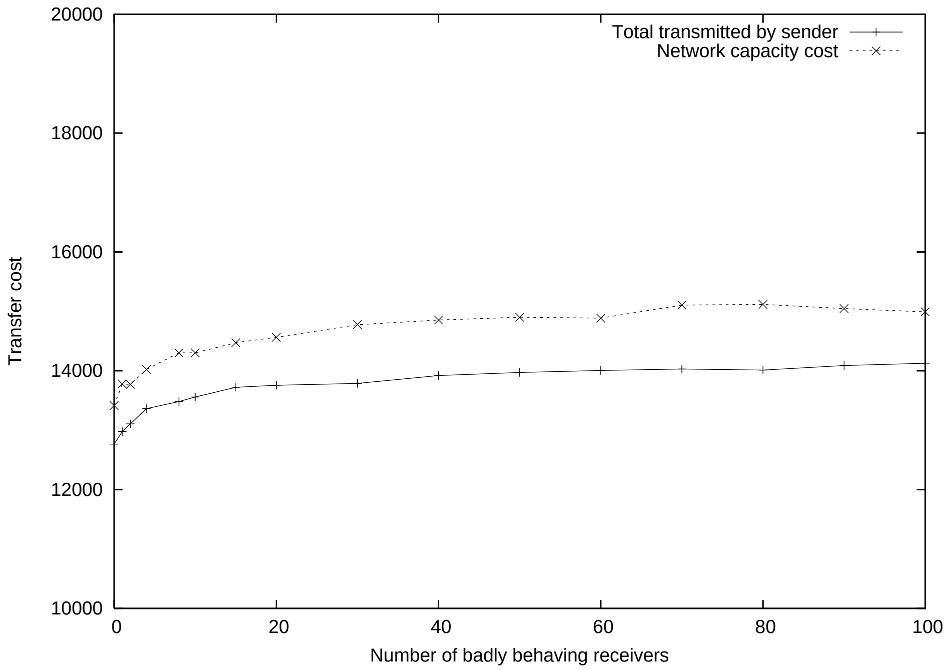

Figure 36: Transfer costs as function of number of badly behaving receivers.

number of receivers in group A. The rest of the 100 receivers receive their packets according to the error rate of group B.

The curve shows a gradual increase in transfer cost as the number of members of group A increases. The receivers in group A need far more FEC packets than the receiver of group B. Therefore, the receivers in group B with the more reliable packet delivery virtually always receive enough packets for a block if the number of FEC packets is large enough to satisfy the members of group A.

The group with the smaller packets loss will rarely if ever need to send requests to the multicast cache. The group will receive enough packets for each block because the receivers with the larger error rate maintain a high number of FEC packets.

Figure 36 can be compared with with the earlier Figure 29. If we limit to observe the behavior of the algorithms between the values 0 and 100 on the X-axis, the curves show a high degree of similarity. Essentially the only difference between the simulations are the existence of additional receivers in one of the simulations. Since the additional receivers never need to send any request packets to the multicast caches, their existence does not have any effect on the multicast cache.

If a multicast cache is transferring content to a large number of receivers, only the receivers with the highest error rate will need to send requests to the multicast cache.

9.10 Delay

In this discussion, delay is defined as the time between when the multicast cache starts sending packets in a block and when a receiver is able to reconstruct the block. A receiver is able to reconstruct a block once it has received

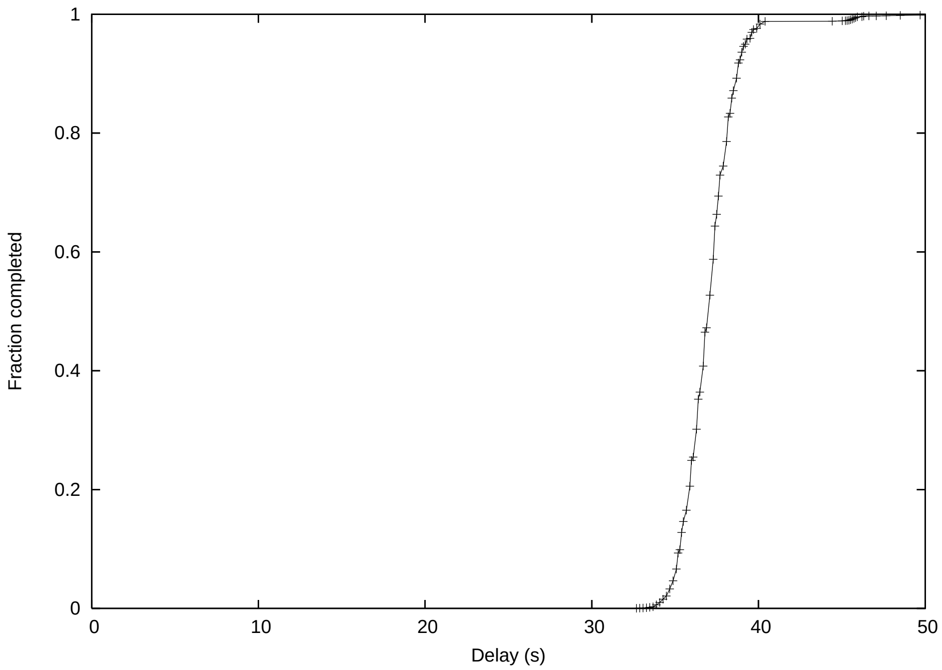

Figure 37: Transfer delay for block with 200 packets.

a sufficient number of packets from the block.

Block size

Figures 37, 38 and 39 illustrate the delay with 3 different block sizes. In the figures, the graph shows the percentage of receivers that have reconstructed the block as the function of time since the multicast cache started transferring the block to its receivers.

The time it would take to transfer a block of data assuming a error rate of 0% is called the *theoretical transmission time*. In the simulations, the packets are sent at a rate of 5 packets/s from the multicast cache to the receivers. For a block consisting of 100 packets, the theoretical transmission time for the block is 100 packets / 5 packets/second = 20 seconds. The FEC packets for each block are sent in parallel with the content packets, starting at the time when the corresponding block starts. It is therefore possible for a receiver to complete its block even before the 20 second time.

Figure 37 illustrates delay for the transmission of a block consisting of 200 packets. Without FEC and with zero errors, the theoretical completion time for the block would be 200 packets / 5 packets/second = 40 seconds. However in the figure, the first receivers complete their block soon after 30 seconds. At the theoretical completion time, more than 97% of the receivers have reconstructed their block. The rest of the receivers will receive their missing data only once they have requested it to be retransmitted by the multicast cache. All of the receivers have received the necessary packets for the block before the 50 second mark.

Figure 38 illustrates the delays for a block size of 100 packets. All other variables have the same values as in the previous simulation. Since block in this simulation consists of 50% less packets than in the previous example, the theoretical completion time for the block is also 50% smaller, i.e., 20

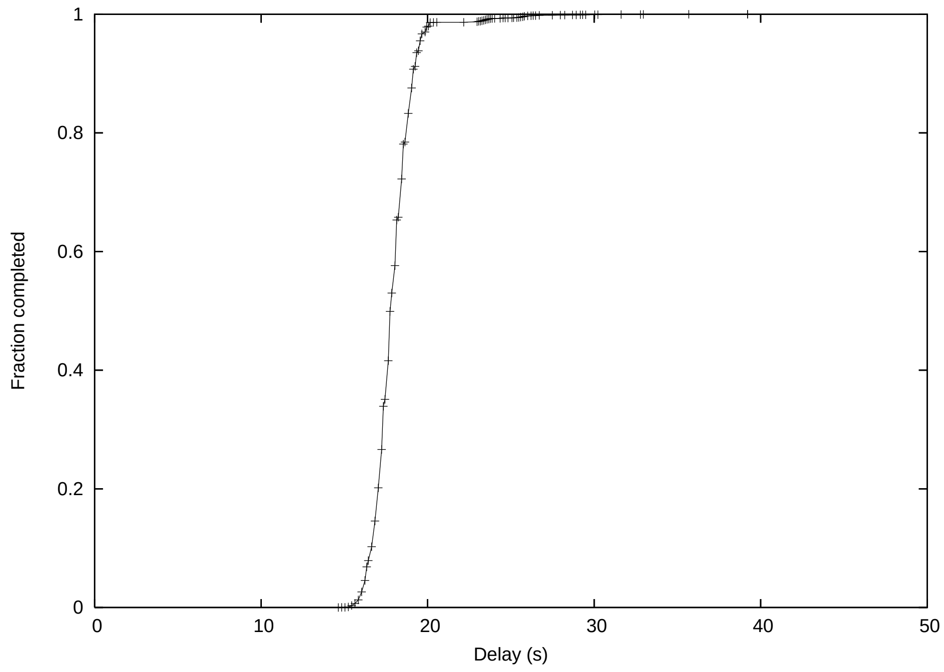

Figure 38: Transfer delay for block with 100 packets.

seconds. Otherwise, the simulation run shows similar behavior as the run with blocks of 200 packets.

Figure 39 illustrates the simulation with the number of packets in a block set to 10. In this case, the fraction of receivers which are able to reconstruct blocks without retransmissions is less than 90%. Figure 40 shows the density function for the number of FEC packets per block during the simulation. The optimal number of FEC packets per block is very small because of the small block size. The value for the number of FEC packets also varies greatly during the simulation because of the autofec adjustment algorithm. The smallest decrease that can be made to the number of FEC packets is to decrease their number with one. However, if the original number of FEC packets was 5, removing one packet amounts to a reduction of 20% of the FEC packets. When the new value for the number of FEC packets is used in sending the next block, a large number of receivers will not receive the required number of packets.

The small block size results in oscillation where the number of FEC packets varies between 0 and 9 as shown in Figure 40. For a large portion of time the number of FEC packets is smaller than the minimum number needed for reliable transmission. The result of the oscillation is, that a significant number of receivers will need retransmissions to receive a sufficient number of packets from each block.

Limited FEC

All of the previous delay curves were computed assuming an unlimited number of FEC packets available to the multicast cache. Therefore the multicast cache was always able to send new FEC packets that have never before been sent to the receivers. In this section, we assume that the number of FEC packets is limited and consider its effects on the delay.

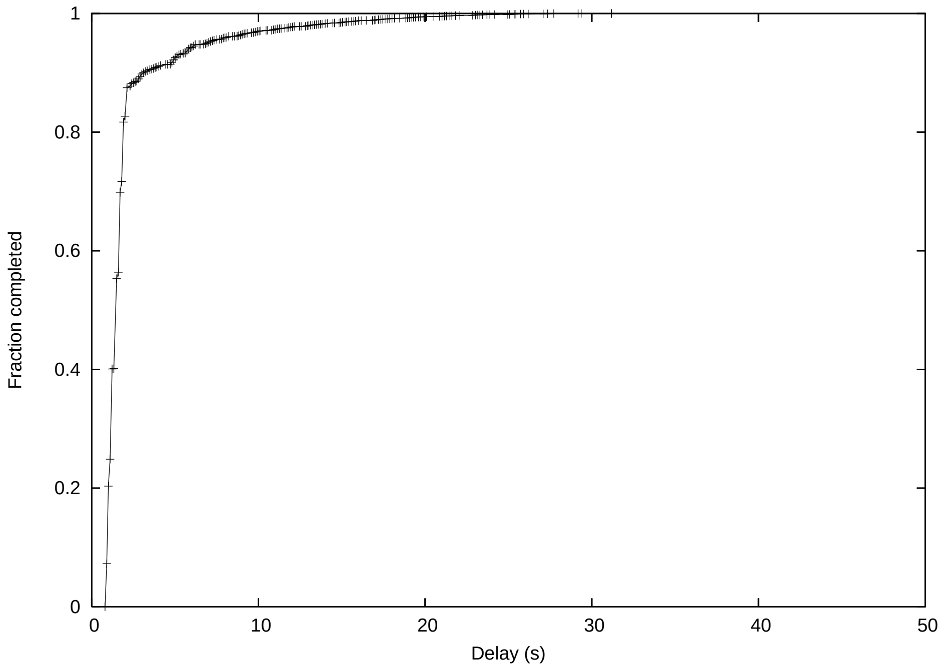

Figure 39: Transfer delay for block with 10 packets.

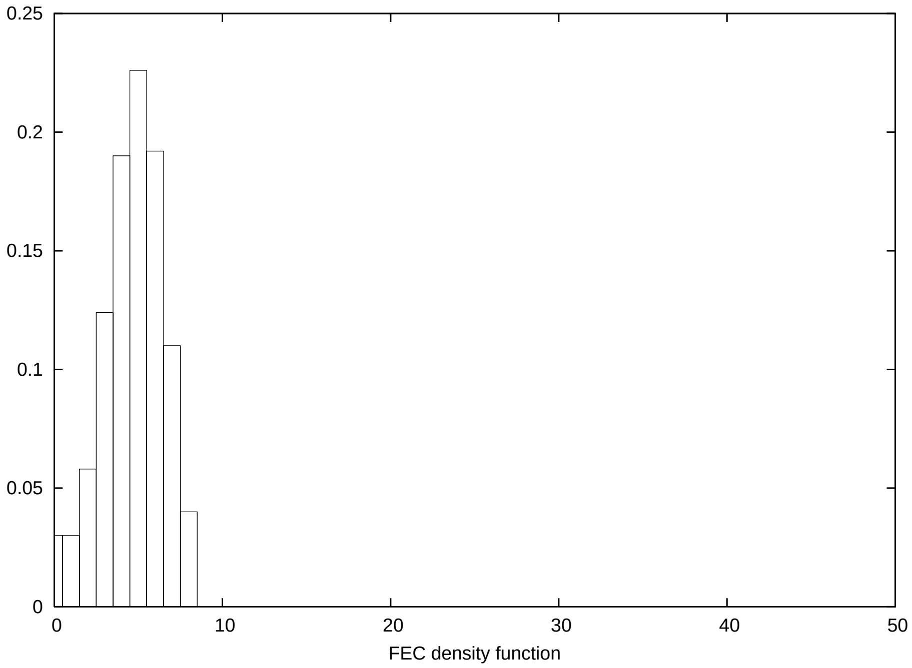

Figure 40: FEC distribution with block size 10 and error rate 15%.

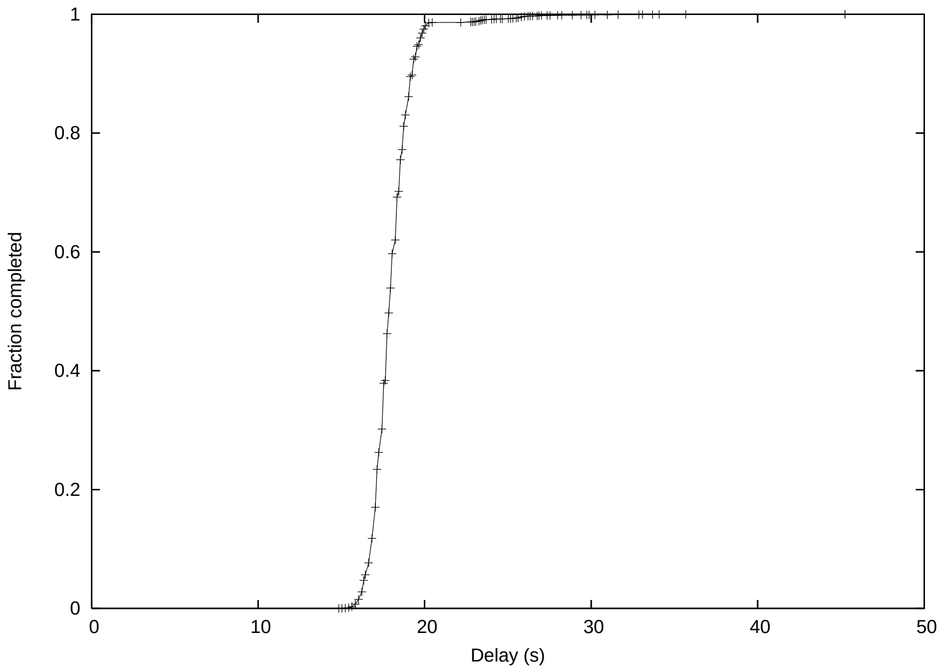

Figure 41: Transfer delay for maximum of 30 FEC packets

Figure 41 illustrates the delay curve when the number of FEC packets is limited to 30 packets per block. The resulting curve is almost indistinguishable from the delay curve in Figure 38, which was computed without any limitations to the number of FEC packets, but with otherwise the same settings. The similarity of the curves show, that the maximum of 30 FEC packets per block are sufficient under the simulated conditions.

Figure 42 illustrates the delay curve when the number of FEC packets has been limited to 20 packets. In this case the number of FEC packets is not fully sufficient to compensate for the packet loss. On average, only 80% of receivers will receive the blocks within the theoretical completion time of 20 seconds.

Retransmission requests are sent by the receivers immediately after the block ends at 20 the second point. The curve continues to rise in response to retransmissions sent by the multicast cache. However, the curve continues to rise at a much lower rate even though retransmissions are sent at a rate of 10 packets/second. The low effectiveness of the retransmissions is caused by a number of factors.

- The retransmitted packets are packets that have been already sent. About 85% of the retransmitted packets are useless to a receiver which has already received a copy of the packet.
- 15% of the retransmitted packets are dropped.
- The FEC packets of the next block share the network capacity used by the retransmissions. The FEC packets of the next block have priority over the retransmissions, and no retransmissions are sent before the FEC packets are sent.

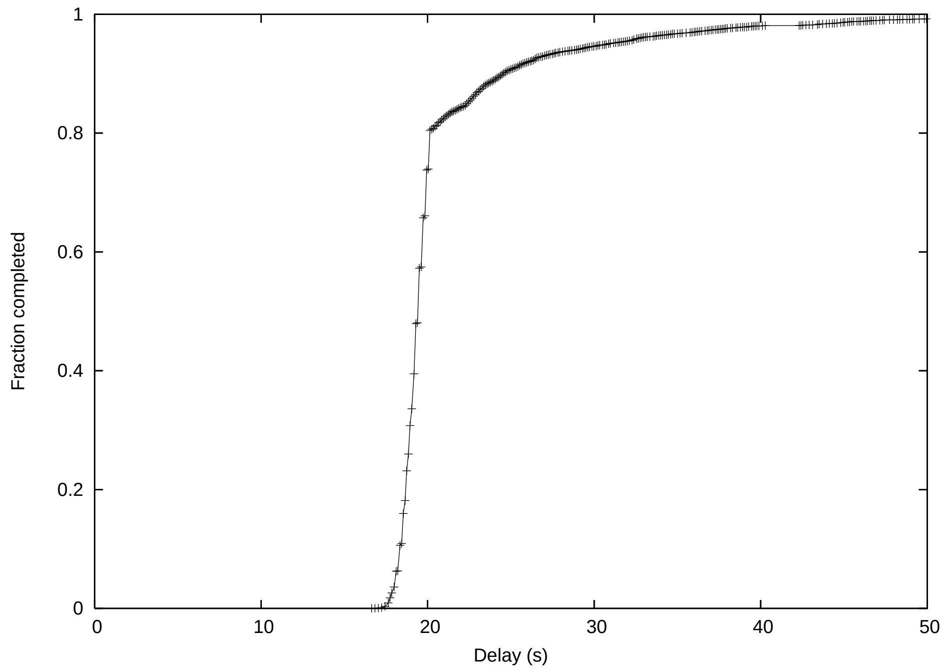

Figure 42: Transfer delay for maximum of 20 FEC packets

Figures 43 and 44 show the behavior of the algorithm when the number of FEC packets is set to 15 and 10 per block, respectively. In both cases, the behavior matches closely the behavior with 20 FEC packets. However, the less FEC packets are available, the longer the blocks take to complete. Also the number of receivers which can be satisfied without retransmissions drops as the number of FEC packets drops.

Packet loss

Figures 45 - 47 illustrate the delay curves for several different packet loss rates. In Figure 45 the loss rate is set at 5%. The result is an S-shaped curve which reaches a value close to 1 at the 20 second point. In some cases, the FEC sent automatically by the multicast cache is not enough to satisfy all receivers. Some receivers receive retransmissions up to 35 seconds after the block was started.

Figures 46 and 47 illustrate the delay curves for 10% and 20% error rates, respectively. Figure 48 combines the three delay curves into one picture. In all cases, the result is an S-shaped curve whose width varies depending on the error rate. Larger error rates result in wider S-curves. In particular, if the error rate is 0 and no FEC packets are sent, the S curve degenerates into a vertical line at the 20 second point. Regardless of the error rate, the curve reaches approximately the 97% satisfaction rate at the 20 second point.

The minimum delay needed to satisfy a small fraction of receivers is the smallest when the error rate in the simulation is the highest. This apparent contradiction is explained by the higher number of FEC packets that are sent when the error rate is higher. In this case, the receivers which experience less packet loss than average complete their block well in advance of the theoretical completion time.

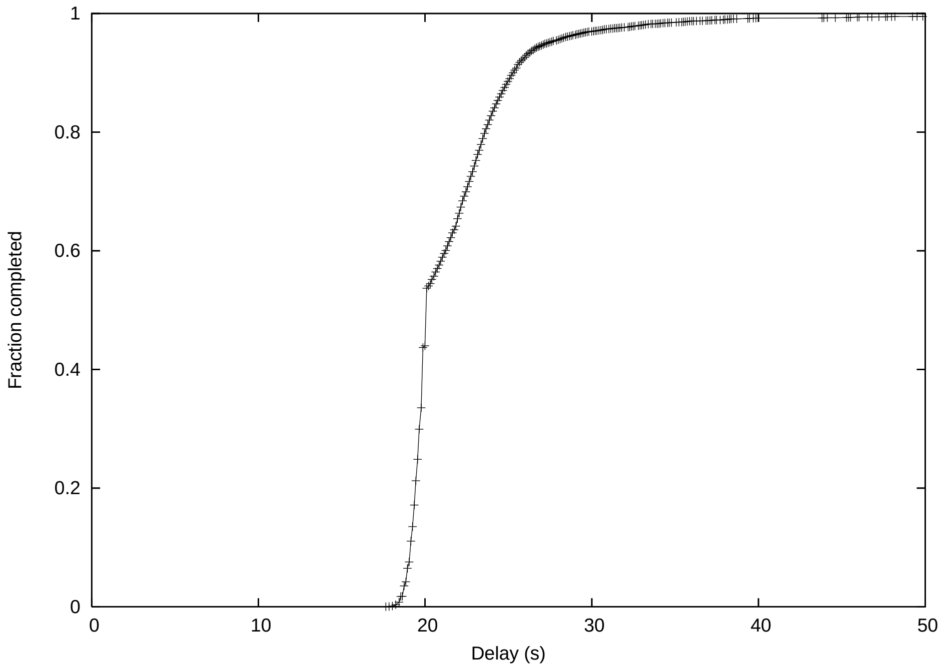

Figure 43: Transfer delay for maximum of 15 FEC packets

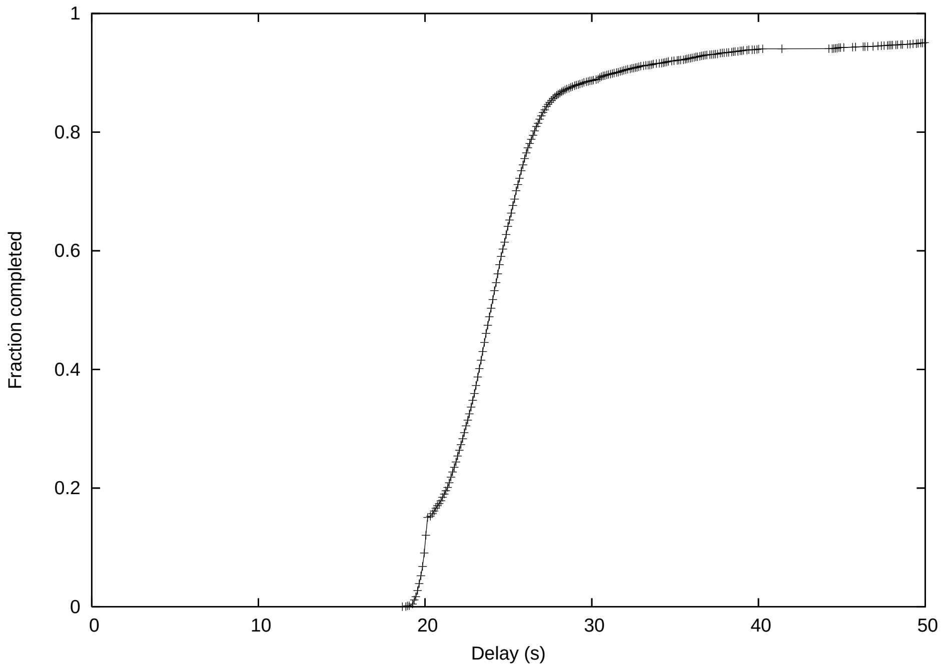

Figure 44: Transfer delay for maximum of 10 FEC packets

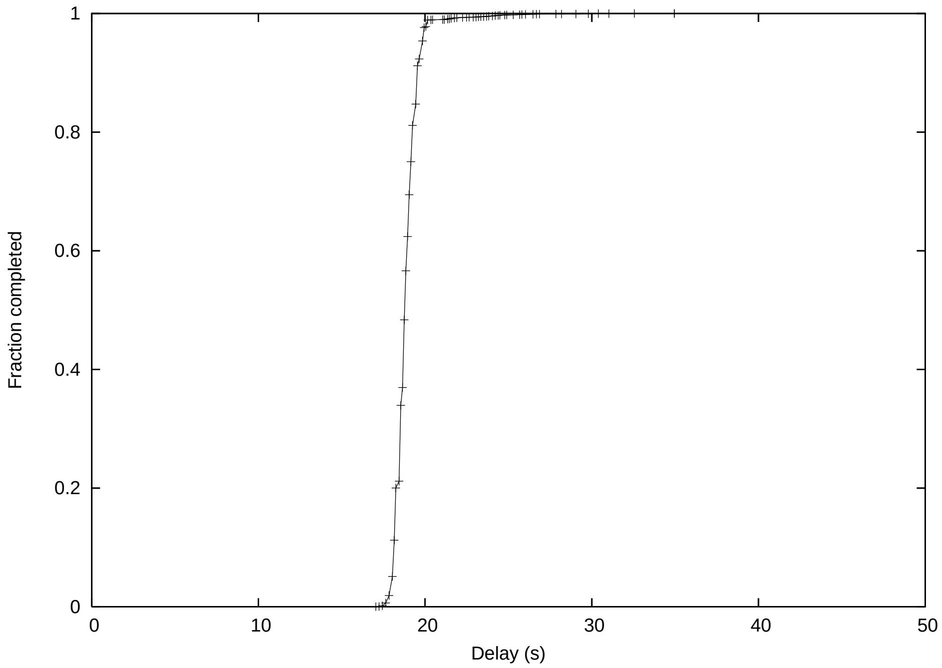

Figure 45: Delay for 5% error probability

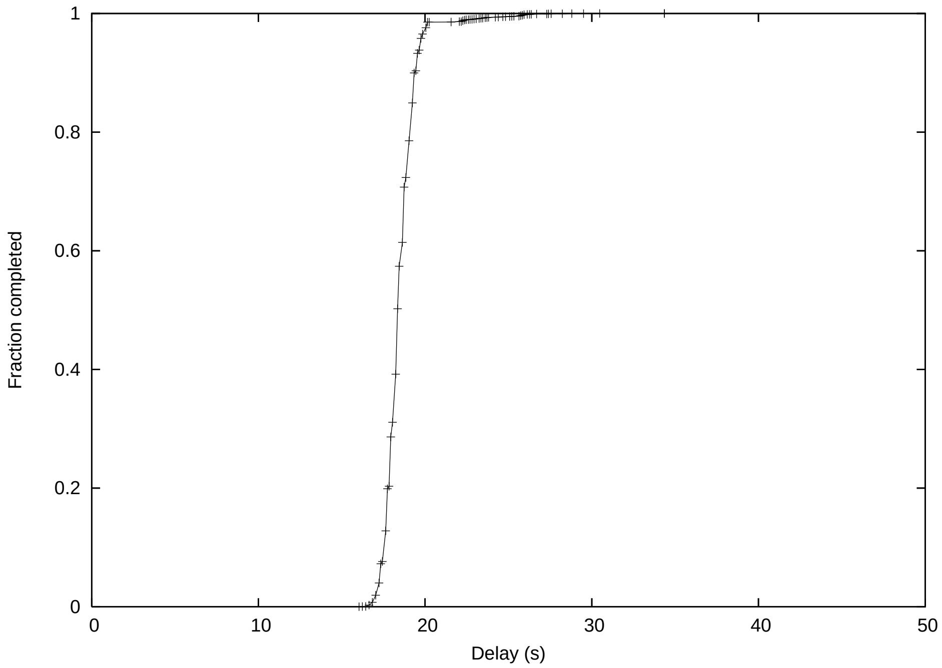

Figure 46: Delay for 10% error probability

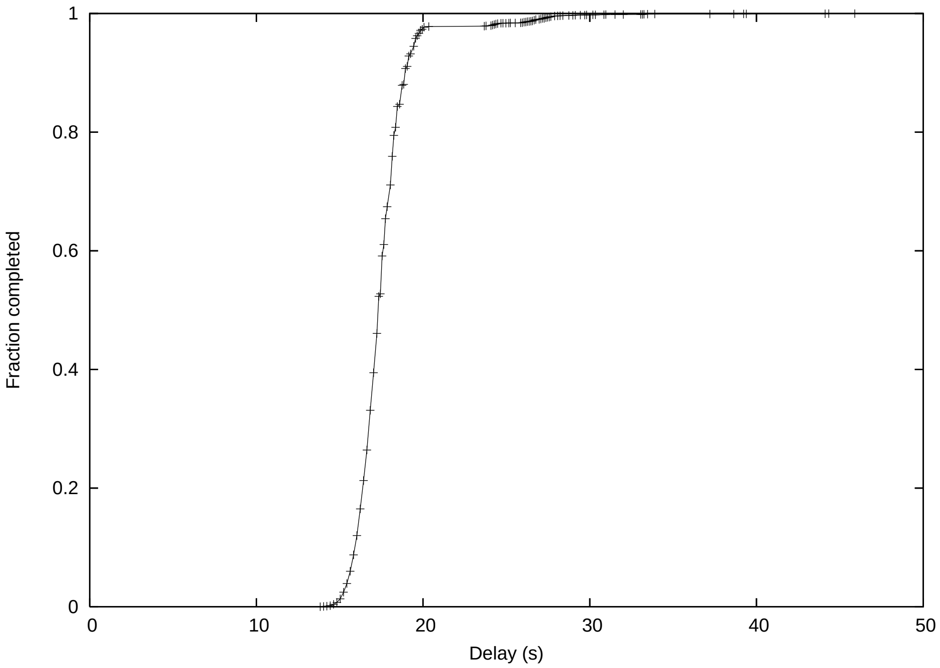

Figure 47: Delay for 20% error probability

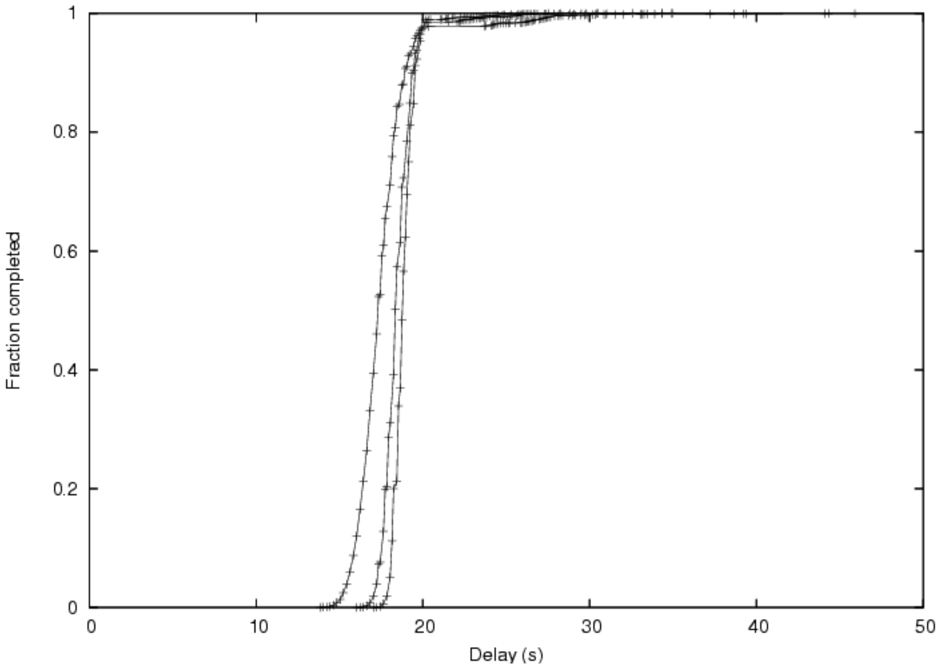

Figure 48: Combined figure of the 5%, 10% and 20% delay curves

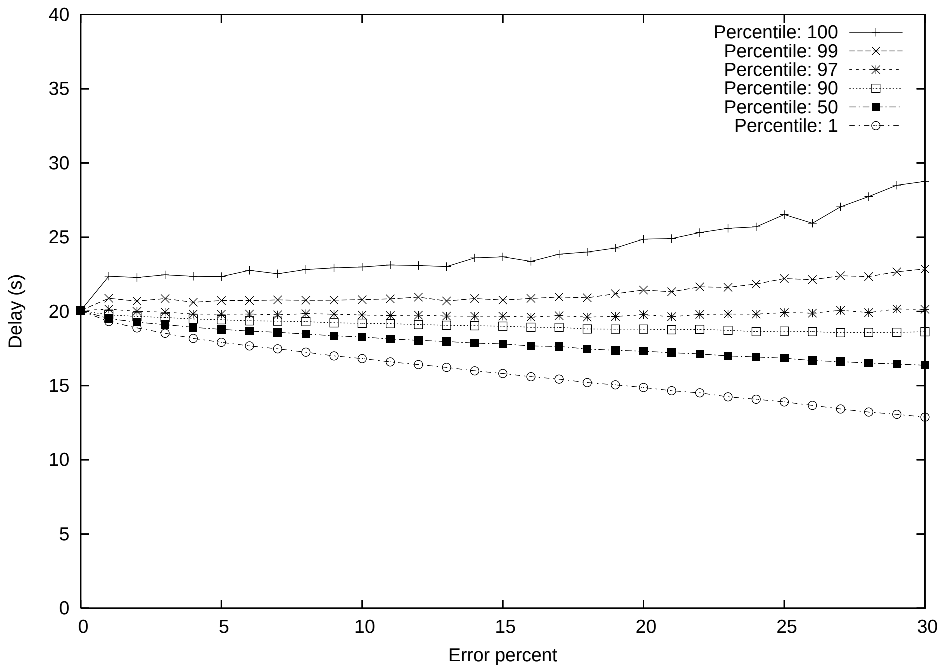

Figure 49: Average completion delay for some percentiles of receivers.

Block delay

Figure 49 gives an alternative view to the delays in completion of blocks. The curves in the figure show average delay percentiles for 100 receivers as the function of the error rate.

The average delay for completing blocks remains below the theoretical limit of 20 seconds for 97% of the receivers up to an error rate of 25%. Even when the error rate exceeds 25%, the increase in delay is very small. The remaining 3% of receivers complete their blocks with a delay little greater than the theoretical delay. For the last 1%, the delay grows up to nearly 30 seconds, which is corresponds to a 50% increase in the delay over the theoretical completion delay.

For the 90th and smaller percentiles, the completion delay decreases as the error rate grows. The same behavior was also seen in Figures 45, 46 and 47, where the delay curves were widened when the error rate was increased. Again, the effect can be explained by the adaptively increased number of FEC packets when the error rate is high. Some receivers are able to complete their blocks early if the number of packets they did not receive was smaller than average.

Preemptive fast requests

The default behavior for a receiver is to send the fast request once it detects that the number of packets it expects to receive is less than the number that it needs. That is, the receiver adds the number of packets already received with the number of packets still expected. If this value is less than the block width, a fast request is sent.

The delay can be further reduced by modifying the fast request algorithm in the receivers. Figure 50 illustrates the delay percentiles with preemption value of 4. That is, a fast request is sent when less than 4 packets more than

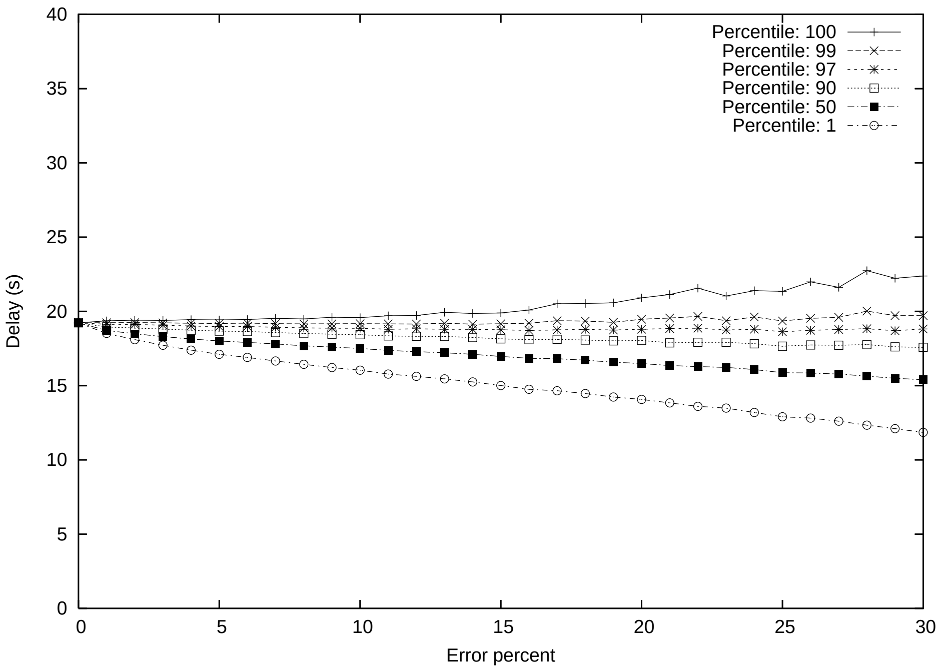

Figure 50: Average completion delay for some percentiles of receivers with preemption value 4.

needed are expected to be received from the multicast cache. Preemption value of 0 corresponds to the receiver behavior without preemption.

In Figure 50, the completion delay for even the 99% curve stays st the 20 second line or below. The slowest 1% of receivers still require some retransmissions, and the curve stays above the 20 second line. By increasing the preemption value even further, even the curve for the slowest 1% can be pushed below the 20 second line. That, however, requires using even more FEC packets, which will increase the transfer cost.

9.11 FEC

Figures 51 - 54 illustrate the number of FEC packets sent from the multicast cache to the receivers as a density function. In each figure the simulation was run using a different error rate. The Figures 51, 52, 53 and 54 show the FEC per block distributions for error rates 5%, 15%, 25% and 35%, respectively.

Each FEC density function shows one or more very low bars well below the large concentration of bars. These anomalies are caused by the FEC adjustment algorithm. The first block in each simulation starts with 0 FEC packets sent in the block. The FEC algorithm converges to its proper values within 1-5 blocks. However, the time before stabilization is visible in the figures.

9.12 Varying error rate

For now, every simulation has been run with a constant error rate during one simulation. Figure 55 shows a different view of a simulation where the error rates vary between several values over time.

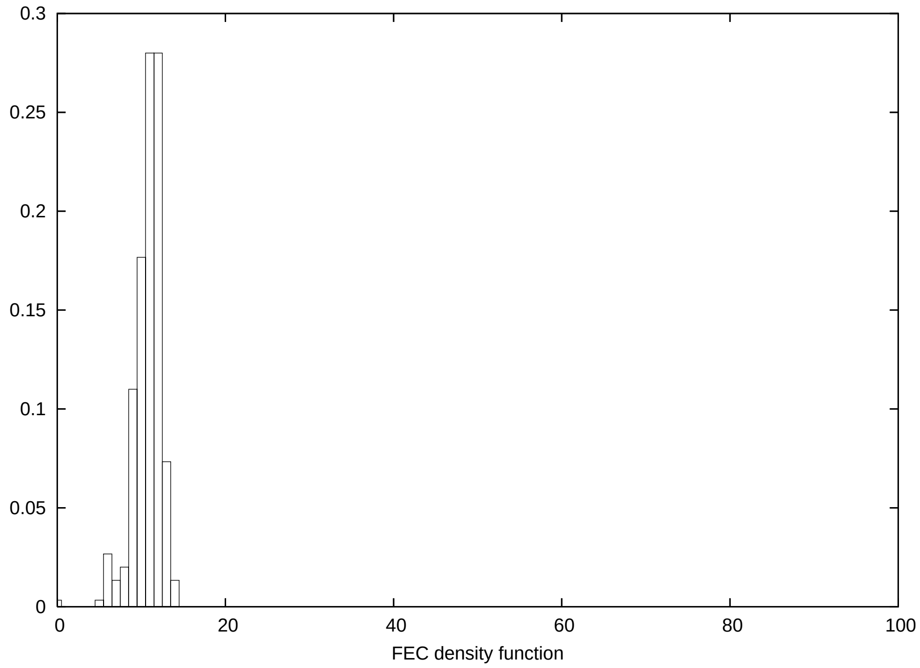

Figure 51: FEC density function for 5% error rate.

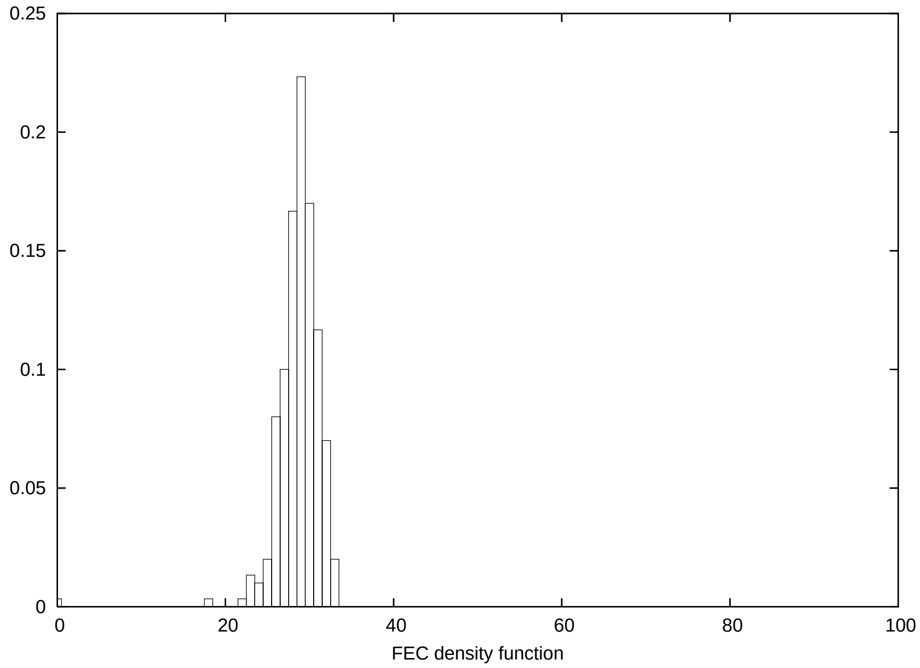

Figure 52: FEC density function for 15% error rate.

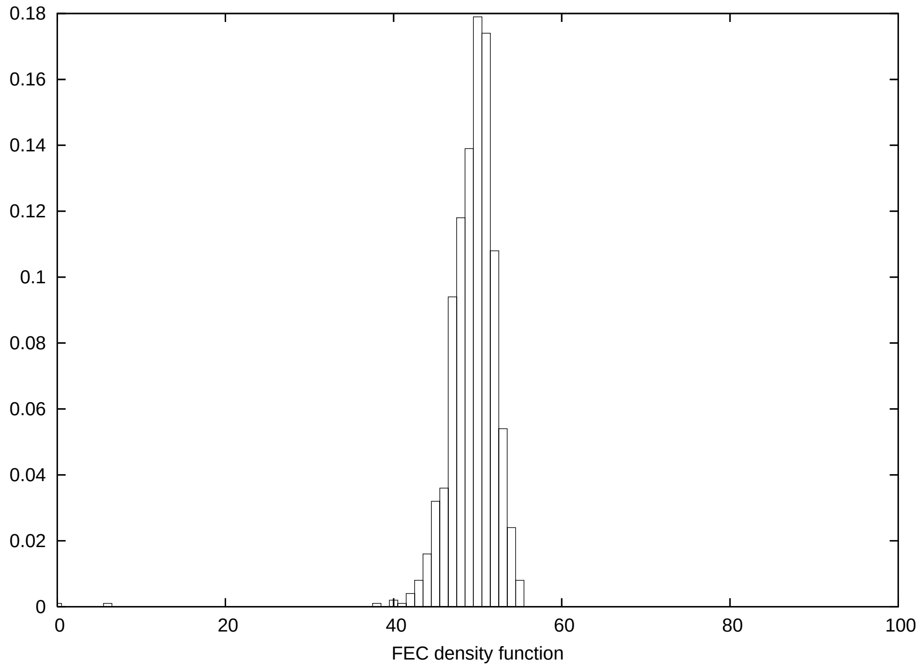

Figure 53: FEC density function for 25% error rate.

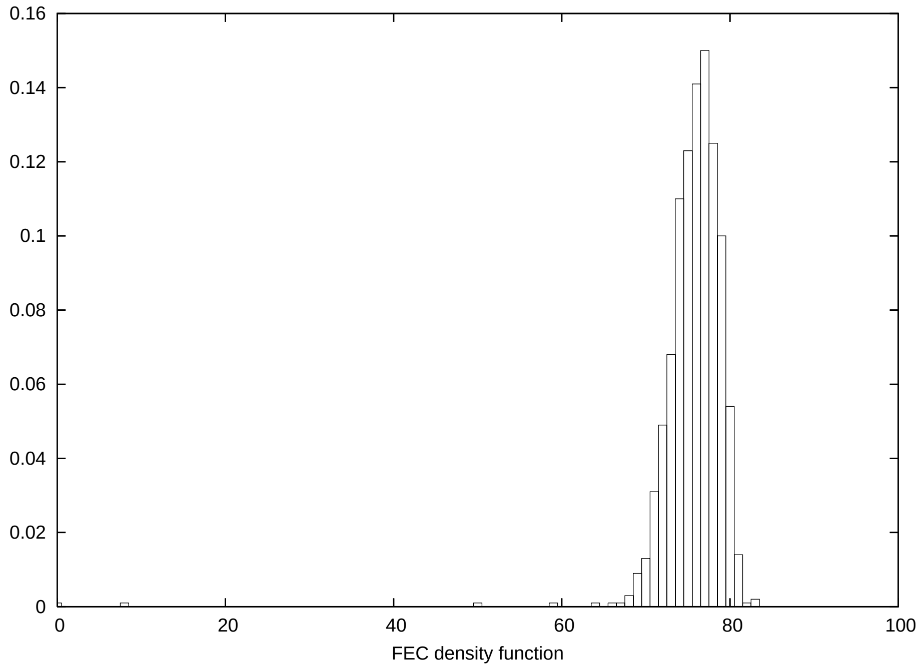

Figure 54: FEC density function for 35% error rate.

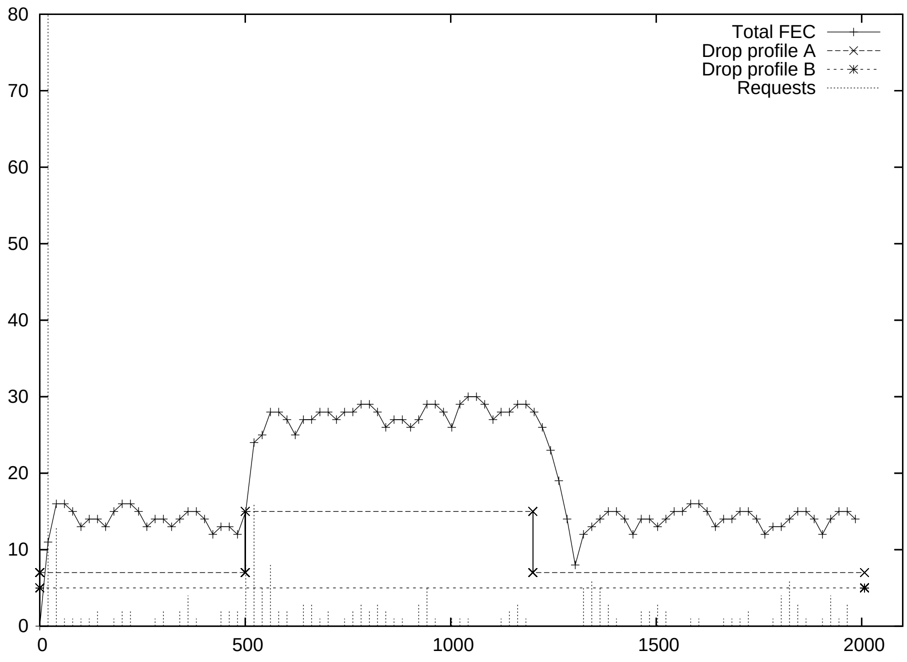

Figure 55: Algorithm behavior for 1000 receivers with varying error rates.

In the figure, the multicast cache transfers content to 1000 receivers. The receivers are divided into two groups with different error rate behaviors. The receivers in group A follow the the drop profile A. This group consists of 50 receivers, which start receiving packets at an error rate of 7%. At 500 seconds of simulation time, the error rate of these receivers increases to 15%. At 1200 seconds, the error rate of theses receivers returns back to 7%. The 950 receivers in group B follow the drop profile B, where the error rate remains constant at 5% during the entire simulation.

The figure also shows the number of fast requests and requests that were sent by the group of receivers. Finally, the figure shows the number of FEC packets per block that was sent for each block.

When the transmission starts, the multicast cache starts sending packets with 0 FEC packets per block. The receivers respond with fast request messages, and within two blocks the number of FEC packets per block has risen to its proper value. After the number of FEC packets has stabilized, the number of request messages from receivers drops.

At 500 seconds, the error rate for group A increases to 15%. The increase is followed by several requests which increase the number of FEC packets per block to a higher value. The packets that were lost before the FEC was increased are also received through retransmissions. Later, at 1200 seconds, the error rate for group A returns to a value of 7%, and the number of FEC packets adjusts to the lower error rate.

Figure 56 gives another example with 100 receivers. Of the receivers, 5 behave according to drop profile A, and the 95 other receivers behave according to drop profile B. In this example, the drop profile for A includes a period of 100 seconds when the drop rate of the group A increases to 100%. This simulates, for example, a break in connectivity when the receivers temporarily move to an area without network connectivity. When the connectivity to

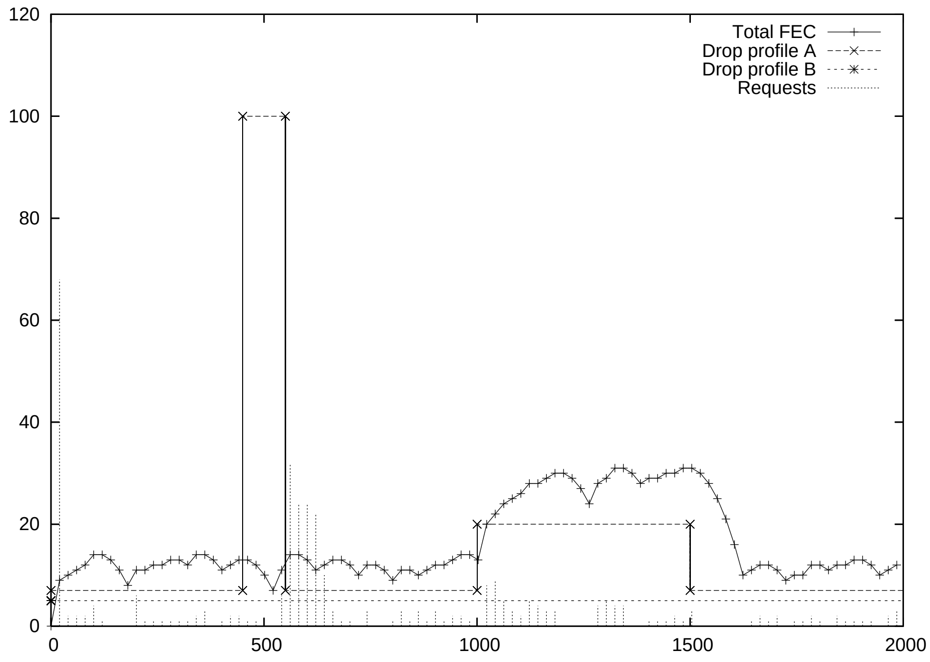

Figure 56: Algorithm behavior for 100 receivers with varying error rates.

the multicast cache is restored, the receivers request the missing packets.

The results show that the protocol can recover from situations where the network connectivity is temporarily lost or significantly degraded.

9.13 Transfer delay with limited FEC

The additional transfer cost from inadequate number of FEC packets was discussed in Section 9.9. In addition to transfer cost, too few FEC packets also increases the delay in transferring blocks.

Figure 57 illustrates percentiles for the average block delay as the function of packet loss rate when the number of FEC packets per block is limited to 20. The delay behavior is similar to the case in Figure 49 with unlimited FEC as long as the error rate remains below 10%. As the error rate increases, a large fraction of the receivers can no longer be satisfied using the FEC packets available in the multicast cache.

If the number of available FEC packets is only a few packets less than needed, the delay behavior for the majority of receivers still remains fairly good. However, as the lack of packets becomes worse, the delay increases rapidly.

If maintaining a small delay in the transmission of blocks is necessary, it is important to have enough FEC packets available in the multicast cache. The multicast cache can then adjust the number of FEC packets to a suitable level.

9.14 Simulation conclusion

The algorithms are able to adapt to a variety of changes in their operating environment. The content provider only needs to configure some basic pa-

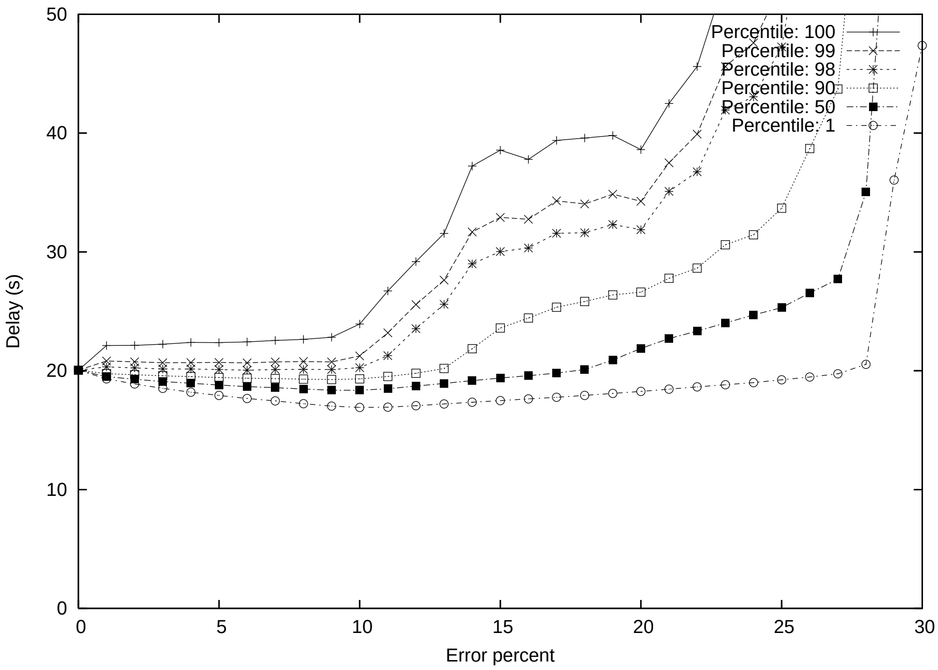

Figure 57: Average delay percentiles for with limited FEC packets per block.

rameters, such as the block size, which cannot be modified by the algorithm. Once the content provider has chosen suitable values for the stream, the algorithms adapt settings which are appropriate for the conditions at each access point. Even if the conditions at an access point are highly unoptimal, the algorithm is able to transfer the content to the receivers.

The most important parameters which need to be configured for efficient multicast delivery are:

- **Block size**, u, controls how many packets of content the server needs to have available before the packets can be sent to the network. Thus, increasing block size also increases delay. If the transferred content has no delay requirements, the block size can be configured to have a large value since transfer costs decrease as the block size increases. However, using values larger than 200 for block size has very little additional advantage, as seen in the simulations.

  Small block values are needed if the transferred content has strict delay requirements. If the delay between the creation and transfer of content needs to be smaller than t seconds, the block size u must be configured to a value which ensures that u packets of content are always created in less than t seconds.

- **FEC packets per block**, f, controls how many packets of forward error correction are created for each block of u packets. Large values for f ensures that the adaptive FEC algorithm and retransmissions have enough FEC packets available to perform efficiently. If too few FEC packets are available, the multicast caches must resend packets that have already been sent, which is harmful for efficiency.

  On the other hand, if the server creates more FEC packets than are necessary, the extra FEC packets use capacity from the core network when they are sent from the server to the multicast caches. Thus, it is necessary to limit the number of FEC packets that are generated even if the unnecessary FEC packets are never sent over the wireless interface.

If unsuitable values are chosen for the parameters, the system can still operate, although the delay requirements may not be met and an unnecessary amount of resources may be used from the wireless access network. However, the system will in most cases still be able to deliver the content.

The architecture is able to deliver content efficiently even to a very large group of receivers even when the error rate in the access network is high or varies.

10 ENERGY CONSUMPTION

The architecture adds some additional data into packets that are used for verifying the authenticity of the content. The processing of this authentication data can potentially consume a significant amount of energy if the receiver is a battery powered mobile device. Three of the most significant consumers of energy are the following:

- **Public key computations.** Packets in the architecture are authenticated by public key signatures. Every signature verification consumes a significant amount of energy.
- **Wireless reception.** Authentication data is carried in every content packet that is delivered in the architecture. The increased length of packets consumes extra energy.
- **Retransmission request.** The energy needed to send the retransmission requests when a sufficient number of packets are not received.

10.1 Energy estimates

We estimate the amount of additional energy that is needed to perform the authentication. The additional energy is estimated as the extra energy per transferred payload. In this computation, payload is defined as the entire packet, except the authentication hashes and the RSA signatures. The computations include the following consumers of energy:

1. Verification of signatures.
2. The energy consumption of receiving the signatures and hashes over the wireless interface.
3. The decreased amount of payload in each packet. The amount of payload in each packet is smaller because the authentication data is included

The energy for sending retransmission requests is not included in the estimates as the number of packets that are sent by a receiver is very small, as was shown in Figure 31.

Next, an equation for estimating the extra energy consumption is created. In these computations, an error rate of 0% is assumed. Table 3 gives the variables used in the equations and their default values. The amount of payload in a packet is counted by subtracting the space consumed by the authentication hashes and the RSA signatures from the MTU.

$$P_{pl} = P - P_{hash} - P_{rsa} = P - H_l \lceil \log_2 w \rceil - P_{rsa}$$

Total energy needed to transfer N bytes of content is

$$E_{totRsa} = \frac{N}{P_{pl}} E_{packet} + \frac{N}{P_{pl}} E_{rx} P + \frac{N}{P_{pl}} \frac{E_{rsa}}{w}$$

Table 3: Variables used in the energy consumption estimates.

Symbol	Description	Default value
w	Hash tree width. (packets)	N/A
E_{rsa}	Energy consumed by one RSA verification.	15.97mJ
E_{rx}	Extra energy needed to receive 1 byte. (J/B)	N/A
H_l	Length of one hash.	16B
P	Network MTU.	1500B
P_{rsa}	Length of one RSA signature.	128B
P_{hash}	Total length of hashes in one packet. (B)	N/A
P_{pl}	Total payload in one packet. (B)	N/A
E_{noAuth}	Total reception energy without authentication. (J)	N/A
E_{totRsa}	Total energy for reception and authentication. (J)	N/A
E_{auth}	Total authentication energy. (J)	N/A
E_{packet}	Energy to start receiving packet. (J)	N/A

$$= \frac{N}{P_{pl}}(E_{packet} + E_{rx}P + \frac{E_{rsa}}{w}) \tag{1}$$

The energy for the case without authentication can be computed by setting $P_{pl} = P$ and $E_{rsa} = 0$ in Equation 1. The equation can then be written as

$$E_{noAuth} = \frac{N}{P}(E_{packet} + E_{rx}P) \tag{2}$$

The cost of authentication is the difference of Equations 1 and 2.

$$E_{auth} = E_{totRsa} - E_{noAuth} \tag{3}$$

10.2 Energy for WLAN and RSA

In [37], the energy consumed by one 1024 bit RSA signature verification was measured to be 15.97mJ. Another paper, [24], gives measurements for IEEE 802.11b WLAN energy consumption. The energy consumption for receiving broadcast data at 2Mbps was measured to be 56μJ initial cost for every packet and 0.50μJ for every additional byte received. The energy consumption for 11Mbps reception was measured as 50μJ for each packet and an additional 0.26μJ for every byte.

The energy cost of an RSA verification is very high compared to the energy cost of receiving a packet. Figure 58 shows the energy costs of authenticating 1MB of content as the function of the hash tree width, w, for WLAN at 2Mbps. Table 4 shows the same information along with values for the total energy for the reception and the cost of reception without any authentication. The difference between the different WLAN types is small as the vast majority of the extra energy is consumed by the RSA verifications.

Energy vs. delay

The authentication energy consumption is highly dependent on the hash tree width, and a value as large as possible for w should be chosen. However, the maximum value for w is limited by the increase in delay that results from

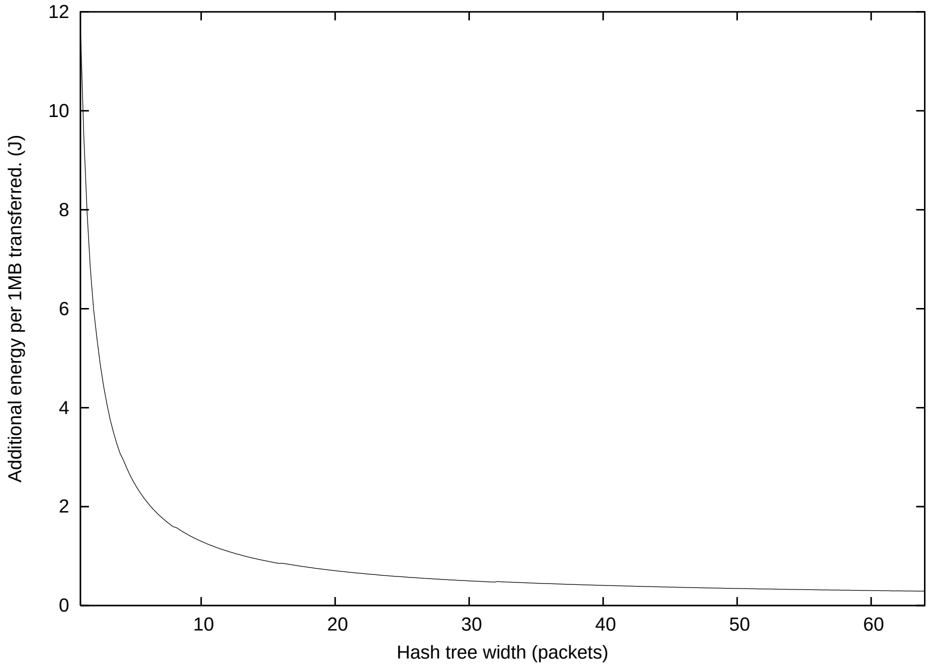

Figure 58: Additional energy needed for authentication as function of hash tree width. (WLAN 2Mbps)

Table 4: Energy consumptions for 1MB content transfer.

w	WLAN 2Mbps			WLAN 11Mbps		
–	E_{totRsa}	E_{noAuth}	E_{auth}	E_{totRsa}	E_{noAuth}	E_{auth}
1	12.0	0.30	11.7	11.9	0.29	11.6
2	6.24	0.30	5.94	6.21	0.29	5.92
4	3.34	0.30	3.04	3.30	0.29	3.01
8	1.88	0.30	1.58	1.84	0.29	1.55
16	1.14	0.30	0.84	1.10	0.29	0.81
32	0.77	0.30	0.47	0.72	0.29	0.43
64	0.59	0.30	0.29	0.54	0.29	0.25
128	0.50	0.30	0.20	0.44	0.29	0.15
256	0.46	0.30	0.16	0.40	0.29	0.11
512	0.44	0.30	0.14	0.38	0.29	0.09
1024	0.44	0.30	0.14	0.37	0.29	0.08
2048	0.44	0.30	0.14	0.37	0.29	0.08
4096	0.45	0.30	0.15	0.37	0.29	0.08
8192	0.46	0.30	0.16	0.38	0.29	0.09

large values of the parameter. The server cannot sign and send its packets until w new packets have been created. That is, if the rate at which content is created by the server is too small compared with the value for w, the delay between content creation and content transfer can become too large. Equation 4 gives the maximum delay, d, that content may need to wait at the server before it can be signed.

$$d = \frac{P_{pl}(w-1)}{S_r} \tag{4}$$

In the equation, S_r is the rate at which content is created in bytes per second. If each packet is signed individually, that is $w = 1$, the delay is reduced to 0. However, such a solution can require an excessive amount of energy for the RSA signature verifications. Table 5 gives some suggestions for compromises between the delay and the energy consumption for different types of content. Bulk data can be any content which can be used only once the entire content has been transferred. Examples of bulk data include software updates. For such content, a large value for w can be used as the only delay criteria is to get the entire content transferred since the incomplete stages of the content are not by themselves useful.

Table 5: Suggestions for hash tree widths for different content types.

Environment:	Network capacity:	Maximum delay:	w
Bulk data	N/A	Large	≥ 128
Live video stream	1Mbps	5s	≥ 128
Live audio stream	128kbps	5s	64
Video conference	1Mbps	1s	64
Conference call	64kbps	1s	4

When delay requirements for the content are needed, the value for w becomes important. However, as long as the ratio between the network capacity requirements and the delay is large, efficient values for w can be used. The energy costs per transferred byte become significantly larger when the delay requirements are strict and the amount of data per second is small. In the case of the 64kbps conference call, only about 6 packets per second are created. To maintain a delay under $1s$, the value for w needs to be less than 6. As the value of w needs to be a power of 2, the largest suitable value for w is 4.

10.3 Energy conclusions

A suitable selection for the hash tree width w can reduce the energy requirements of the authentication to a value which is virtually insignificant. For bulk data with no strict delay requirements, the additional authentication energy can be as small as 0.08J for 1MB of data. If content with low data rates and strict delay requirements are transferred, the energy costs of the transfer become larger but still remain small.

A typical battery for a small handheld device such as a mobile phone has an energy capacity of about 10kJ. If we assume that all of the energy were used only for the authentication of the incoming content, the 10kJ battery would have enough energy to authenticate, for example, the following amounts of content:

- 113 hours of conference calls at 64kbps rate with 1s maximum delay.
- 141 hours of live video at 1024kbps rate with 5s maximum delay.
- 125GB of bulk data with no delay requirements.

Naturally a real mobile device uses most of its energy for tasks other than content authentication. However, the numbers show that the amount of additional energy needed for the authentication is miniscule in comparison with the battery capacities that are currently available in these devices.

11 IMPACTS ON THE WORLD

This chapter discusses the technical changes which are needed or recommended to get the architecture into commercial use. These changes allow businesses to utilize multicast and multichannel delivery to distribute content to a very large group of receivers. The architecture is designed to operate in heterogenous networks and utilize the resources in the network in the most efficient way possible.

The protocols and algorithms have not been designed to operate in just one specific environment but in a wide variety of different networks. For example, if the uplink capacity is especially expensive in some environment, the protocols can be adjusted to minimize the uplink capacity that is needed. In other cases, it may be more important to minimize the amount of traffic within the fixed network.

11.1 Necessary changes

The following software and hardware are necessary to enable multicast delivery using the architecture of this dissertation.

1. The server software which creates authenticated and encrypted multicast packets from the content.

2. The receiver software which receives the content from the network. The software also verifies the authenticity of the content, and delivers the content to applications which use it. The receiver software also includes the security module functionality, if the security module is not implemented in hardware.

3. Multicast cache software which stores and forwards multicast packets. The multicast cache also sends retransmissions in response to requests from receivers.

4. Key management software which is used for requesting content keys for encrypted content.

5. Multicast caches. A suitable number of multicast caches which are able to send FEC packets and retransmissions to the receivers.

None of the necessary changes include modifications in the core infrastructure of the network, such as routers.

11.2 Enhancing changes

The following additional changes can further enhance the efficiency of the architecture. However, these changes are not needed to build a working system.

1. Multicast in the fixed network. The multicast packets can be transferred from the server to the multicast caches over TCP or UDP streams, if necessary. However, multicast can reduce network capacity use in the fixed network too.

2. Unidirectional link support for IGMP. The current IGMP protocol does not allow multicast transmission over a unidirectional last-hop link.

3. Internet protocol (version 4 or 6). A protocol number for the architecture which allows the protocol to run directly over the network layer. The protocol can function over UDP, but the additional UDP header in each content packet is not needed and it is unnecessary overhead.

4. Standardization of the architecture and the protocols. Standardization allows different content providers, multicast caches, receivers and key servers to interoperate.

5. Establishing a TTP which maintains revocation lists and distributes verification keys for the content.

12 CONCLUSION

In this dissertation, an architecture for efficient, secure and reliable delivery of digital content to mobile terminals using heterogenous wireless access networks has been created. A content delivery system needs to protect both the content providers and the users from attacks. The users need the ability to verify that the content they are receiving has been created by a legitimate content provider. The content providers, on the other hand, must be protected against unauthorized use of their content. Content providers must also be protected against attacks on their reputation, where an attacker tries to distribute forged or modified content in the content provider's name.

The created architecture is able to utilize multicast as the underlying distribution technology. The basic Internet multicast architecture was extended with two additional techniques to improve its reliability and scalability. Forward error correction (FEC) is used to improve reliability against random packet loss. If only a few packets are dropped, the architecture can regenerate the lost data using redundancy found in the stream. The other technique used for improving reliability is retransmissions. A mobile terminal, which has not received enough packets, can request the packets to be retransmitted by a local cache. The two reliability techniques together allow the delivery of content such as software updates that require reliable transfer.

The architecture has the ability to receive pieces of content via several alternative untrusted access networks or intermediate caches. Even then, the content can still be source authenticated. The architecture also enables the separation of content transfer from content consumption, which allows a user to receive content whenever it is the most convenient, without yet committing to paying for it.

The architecture scales even to a very large group of receivers. The adaptive FEC algorithm optimizes the operation of the architecture to utilize as little resources from the access network as possible. Simulations show that the adaptive FEC algorithm can be used to handle loss of packets in networks with both static and dynamically changing error rates. The algorithm quickly adapts to changing conditions in the wireless network. Even when too few error correction packets are created in the server, the system is able to deliver the content fairly efficiently through retransmissions.

The additional energy consumption of the architecture is small and it does not limit the implementation of the architecture even in a mobile device which operates on battery power. The architecture can be used for transferring bulk content as well as content with strict delay requirements. The content provider can modify the parameters used in creating the content to optimize between factors such as delay, reliability and network capacity efficiency.

A proof of concept implementation of the architecture for Linux has been built. The implementation is able to operate as a content delivery system for existing applications such as web browsers or video players without any modifications to the applications.

The architecture offers a new mechanism for content providers to distribute their content to a large group of mobile, wireless users. The content providers can maintain their control over the content because the distribution and the consumption of the content have been separated. The content can be transferred to users by means of any technology including multicast, unicast or peer-to-peer. The channel can be chosen based on criteria such as capacity and cost. Content can be controlled based on strong security solutions where only legitimate user may actually use the content.

References

[1] Ålands Datacommunikation Ab, Ålcom. http://www.alcom.aland.fi/ [Linked 10.12.2004].

[2] Hymn. Hear your music anywhere. http://hymn-project.org/ [Linked 22.11.2004].

[3] iTunes Music Store. http://www.apple.com/itunes [Linked 20.4.2006].

[4] Maxisat. http://www.maxisat.fi/ [Linked 22.11.2004].

[5] Suomi Communications. http://www.suomicom.fi/ [Linked 22.11.2004].

[6] TeliaSonera HomeRun. http://www.homerun.telia.com/ [Linked 31.10.2005].

[7] ETSI TR 101 200 v1.1.1, Digital Video Broadcasting (DVB); A guideline for the use of DVB specifications and standards, September 1997.

[8] IEEE 802.11b-1999 Supplement to 802.11-1999, Wireless LAN MAC and PHY specifications: Higher Speed Physical Layer (PHY) in the 2.4GHz band, Sep 1999.

[9] 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 Player (PHY) specifications - Amendment 1: High-speed Physical Layer (PHY) in the 5 GHz band, 1999.

[10] IEEE 802.11g-2003 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-Speed Physical Layer Extensions in the 2.4GHz Band, 2003.

[11] Introduction to IEEE 802.20, Mar 2003.

[12] ETSI EN 302 304 "Transmission System for Handheld Terminals (DVB-H), November 2004.

[13] B. Adamson, C. Bormann, M. Handley, and J. Macker. NACK-Oriented Reliable Multicast Protocol (NORM). Internet Draft, draft-ietf-rmt-pi-norm-10.txt (work in progress), July 2002.

[14] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM SIGCOMM '02*, August 2002.

[15] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrmann. The secure real-time transport protocol (srtp). Request For Comments 3711, IETF, March 2004.

[16] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. Request For Comments 3376, IETF, October 2002.

[17] C. Candolin and H. Kari. An architecture for context aware management. In *Proceedings of IEEE MILCOM 2003*, Boston, Massachusetts, USA, October 2003.

[18] M. Castro, P. Druschel, A.M. Kermarrec, and A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications*, 20:1489–1499, 2002.

[19] D. Cheriton and S. Deering. Host groups: A multicast extension for datagram internetworks. In *Proceedings of the Ninth Data Communications Symposium (ACM/IEEE)*, Sep 1985.

[20] S. Deering. Host Extensions for IP Multicasting. Request For Comments 1112, IETF, August 1989.

[21] S. Deering and D. Cheriton. Host groups: A multicast extension to the internet protocol. Request For Comments 3810, IETF, June 2004.

[22] J. Eggers, R. Buml, R. Tzschoppe, and B. Girod. Scalar costa scheme for information embedding.

[23] C. Eklund, R. Marks, K. Stanwood, and S. Wang. IEEE Standard 802.16: A Technical Overview of the WirelessMAN Air Interface for Broadband Wireless Access. *IEEE Communications Magazine*, pages 98–107, June 2002.

[24] L. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM*, pages 1548–1557, 2001.

[25] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. IETF Request For Comments 3775, Jun 2003.

[26] A. Kayssi, H. Karaki, and W Abu-Khraybeh. RTP-based caching of streaming media. In *8th IEEE International Symposium on Computers and Communication (ISCC'03)*, July 2003.

[27] C. Lin and S. Chang. SARI: self-authentication-and-recovery image watermarking system. In *ACM Multimedia*, pages 628–629, 2001.

[28] C. Lin, M. Wu, J. Bloom, M. Miller, I. Cox, and Y. Lui. Rotation, scale, and translation resilient public watermarking for images, 2000.

[29] M. Luby. LT codes. In *Proceedings of The 43rd Annual IEEE Symposium on Foundations of Computer Science.*, 2002.

[30] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft. Asynchronous layered coding (ALC) protocol instantiation. Request For Comments 3450, IETF, December 2002.

[31] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D.. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, February 2001.

[32] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft. Forward error correction (FEC) building block. Request For Comments 3452, IETF, December 2002.

[33] L. Matheson, S. Mitchell, T. Shamoon, R. Tarjan, and F. Zane. Robustness and security of digital watermarks. In *Financial Cryptography*, pages 227–240, 1998.

[34] R. Merkle. A certified digital signature. *Advances in Cryptology - CRYPTO'87*, pages 369–378, 1987.

[35] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[36] A. Perrig, D. Song, R. Canetti, J. Tygar, and B. Briscoe. Timed efficient stream loss-tolerant authentication (tesla): Multicast source authentication transform introduction. Request For Comments 4082, IETF, June 2005.

[37] N. Potlapally, S. Ravi, A. Raghunathan, and N. Jha. Analyzing the energy consumption of security protocols. In *IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, August 2003.

[38] Y Rao, S.G. Seshan, and S. Zhang. A case for end system multicast. *IEEE Selected Areas in Communications*, 20:1456 – 1471, 2002.

[39] L. Rizzo, G. Iannaccone, L. Vicisano, and M. Mandley. PGMCC single rate multicast congestion control: Protocol Specification. Internet Draft, draft-rmt-bb-pgmcc-03.txt (work in progress), July 2004.

[40] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM Computer Communication Review*, 27(2):24–36, April 1997.

[41] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. Request For Comments 3550, IETF, July 2003.

[42] F. Stevenson. Cryptanalysis of Contents Scrambling System. Available at http://www-2.cs.cmu.edu/~dst/DeCSS/FrankStevenson/ [Linked 12.11.2004].

[43] U. Varshney. Multicast over wireless networks. *Communications of the ACM*, 45:31 – 37, 2002.

[44] R. Vida and L. Costa. Multicast listener discovery version 2 (MLDv2) for IPv6. Request For Comments 3810, IETF, June 2004.

[45] B. Weis. The use of rsa/sha-1 signatures within encapsulating security payload (esp) and authentication header (ah). Request For Comments 4359, IETF, January 2006.

[46] J. Widmer and M. Handley. TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification. Internet Draft, draft-ietf-rmt-bb-tfmcc-03.txt (work in progress), July 2004.

[47] C. Wong and S. Lam. Digital signatures for flows and multicasts. In *IEEE ICNP '98*, 1998.

[48] G. Yu, C. Lu, H. Liao, and J. Sheu. Mean quantization blind watermarking for image authentication, 2000.

A COMPARISON OF SIMULATION VS. MATHEMATICAL MODEL

The complexities in the algorithms makes it difficult to give estimates on the performance of the algorithms using mathematics. However, it is possible to model the protocol mathematically under some specifically chosen circumstances. Below, we compare the results of our simulation to a mathematical model of the protocol.

A.1 Forward error correction

In our test case, we send one block of content consisting of d data packets to a group of m receivers. The number of FEC packets in the model is denoted by f. No retransmissions are performed in this model, but only a probability of success for the transmission is computed. The transmission is considered successful if all m receivers receive at least d packets of the total of $d + f$ packets sent. The variables used in the model are show in Table 6.

Table 6: Variables in the mathematical model.

m	Number of receivers.
d	Number of data packets in one FEC block.
f	Number of FEC packets computed from the d data packets.
p	Probability of successful reception of a packet.
$P_r(m, f)$	Probability that all m receivers receive at least d packets of the $d + f$ packets that are sent.

In the mathematical model, the probability of success is computed using Equation 5. The results computed from the equation are illustrated in Figure 59. The error probability, p, is set to 12%. The same probability computation performed by our simulator is shown in Figure 60. The simulation was run 1000 times for each point in the grid. The difference between the mathematical model and the simulated results are shown in Figure 61.

$$P_r(m, f) = \Big(1 - \sum_{x=1}^{d} \binom{d+f}{x} p^x (1-p)^{d+f-x}\Big)^m \tag{5}$$

The results show that the simulator and the mathematical model give very similar results for this example case.

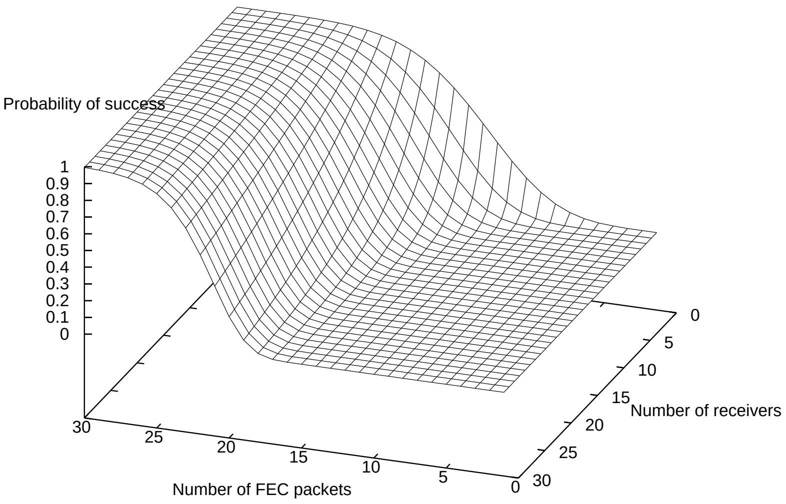

Figure 59: Mathematical probability of successful transfer as the function of the number of listeners and the number of FEC packets per block.

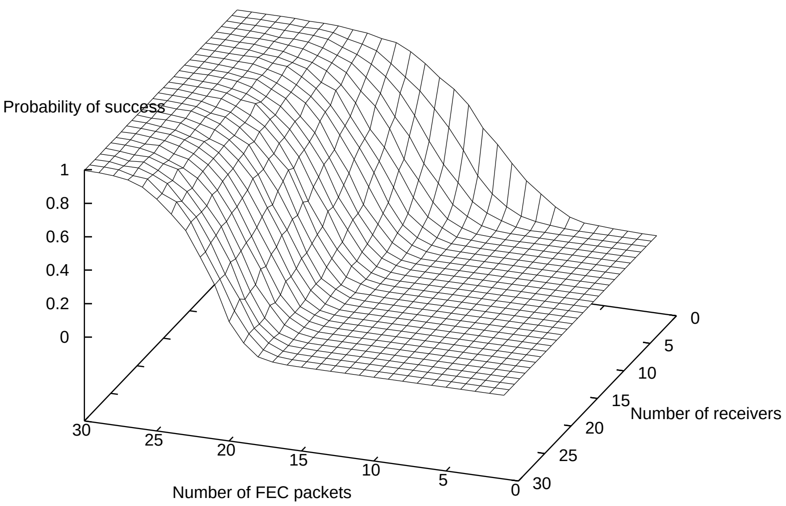

Figure 60: Simulated probability of successful transfer as the function of the number of listeners and the number of FEC packets per block.

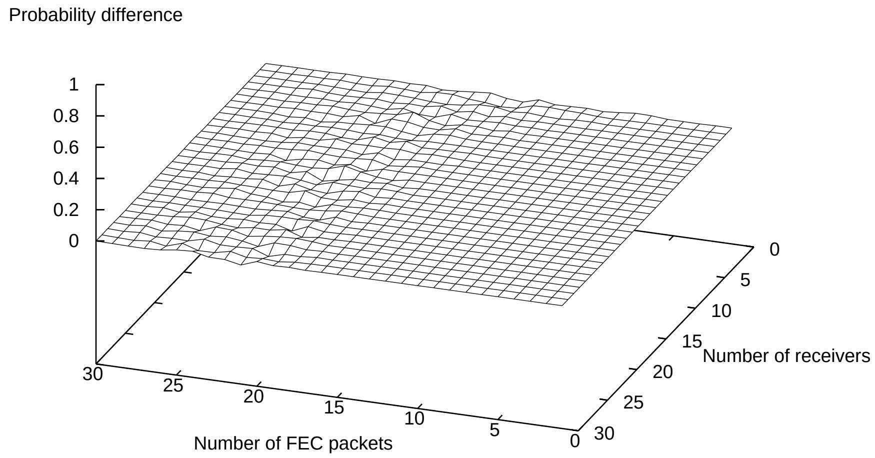

Figure 61: Difference between the mathematical and simulated model.

HELSINKI UNIVERSITY OF TECHNOLOGY LABORATORY FOR THEORETICAL COMPUTER SCIENCE RESEARCH REPORTS

HUT-TCS-A90	Matti Järvisalo	Proof Complexity of Cut-Based Tableaux for Boolean Circuit Satisfiability Checking. March 2004.
HUT-TCS-A91	Mikko Särelä	Measuring the Effects of Mobility on Reactive Ad Hoc Routing Protocols. May 2004.
HUT-TCS-A92	Timo Latvala, Armin Biere, Keijo Heljanko, Tommi Junttila	Simple Bounded LTL Model Checking. July 2004.
HUT-TCS-A93	Tuomo Pyhälä	Specification-Based Test Selection in Formal Conformance Testing. August 2004.
HUT-TCS-A94	Petteri Kaski	Algorithms for Classification of Combinatorial Objects. June 2005.
HUT-TCS-A95	Timo Latvala	Automata-Theoretic and Bounded Model Checking for Linear Temporal Logic. August 2005.
HUT-TCS-A96	Heikki Tauriainen	A Note on the Worst-Case Memory Requirements of Generalized Nested Depth-First Search. September 2005.
HUT-TCS-A97	Toni Jussila	On Bounded Model Checking of Asynchronous Systems. October 2005.
HUT-TCS-A98	Antti Autere	Extensions and Applications of the A^* Algorithm. November 2005.
HUT-TCS-A99	Misa Keinänen	Solving Boolean Equation Systems. November 2005.
HUT-TCS-A100	Antti E. J. Hyvärinen	SATU: A System for Distributed Propositional Satisfiability Checking in Computational Grids. February 2006.
HUT-TCS-A101	Jori Dubrovin	Jumbala — An Action Language for UML State Machines. March 2006.
HUT-TCS-A102	Satu Elisa Schaeffer	Algorithms for Nonuniform Networks. April 2006.
HUT-TCS-A103	Janne Lundberg	A Wireless Multicast Delivery Architecture for Mobile Terminals. May 2006.

ISBN 951-22-8186-4
ISSN 1457-7615