Distribution of Multimedia Streams to Mobile Internet Users

Cristian Hesselman

Telematica Instituut

Enschede, The Netherlands, 2005

Telematica Instituut Fundamental Research Series, No. 014 (TI/FRS/014)

Cover Design: Studio Oude Vrielink, Losser and Jos Hendrix, Groningen
Book Design: Lidwien van de Wijngaert and Henri ter Hofte
Printing: Universal Press, Veenendaal, The Netherlands

Telematica Instituut Fundamental Research Series (also see: http://www.telin.nl/publicaties/frs.htm)

001 G. Henri ter Hofte, *Working apart together: Foundations for component groupware*
002 Peter J.H. Hinssen, *What difference does it make? The use of groupware in small groups*
003 Daan D. Velthausz, *Cost-effective network-based multimedia information retrieval*
004 Lidwien A.M.L. van de Wijngaert, *Matching media: information need and new media choice*
005 Roger H.J. Demkes, *COMET: A comprehensive methodology for supporting telematics investment decisions*
006 Olaf Tettero, *Intrinsic information security: Embedding security issues in the design process of telematics systems*
007 Marike Hettinga, *Understanding evolutionary use of groupware*
008 Aart T. van Halteren, *Towards an adaptable QoS aware middleware for distributed objects*
009 Maarten Wegdam, *Dynamic reconfiguration and load distribution in component middleware*
010 Ingrid J. Mulder, *Understanding designers, designing for understanding*
011 Robert J. Slagter, *Dynamic groupware services: Modular design of tailorable groupware*
012 Nikolay K. Diakov, *Monitoring distributed object and component communication*
013 Cheun Ngen Chong, *Experiments in rights control expression and enforcement*

Samenstelling promotiecommissie:

Voorzitter:	prof.dr.ir. H. Brinksma (Universiteit Twente)
Secretaris:	prof.dr.ir. A.J. Mouthaan (Universiteit Twente)
Promotor:	prof.dr.ir. E. Huizer (Universiteit Twente)
Assistent promotoren:	dr.ir. E.H. Eertink (Telematica Instituut)
	dr.ir. I.A. Widya (Universiteit Twente)
Leden:	prof.dr. F.M.T. Brazier (Vrije Universiteit Amsterdam)
	prof.dr.ir. B.R.H.M. Haverkort (Universiteit Twente)
	prof.dr.ir. I.G.M.M. Niemegeers (Technische Universiteit Delft)
	prof.dr.ir. L.J.M. Nieuwenhuis (Universiteit Twente)
	prof.dr. H. Schulzrinne (Columbia University, New York, USA)

ISSN 1388-1795; No. 014
ISBN 90-75176-70-8

Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
E-mail: info@telin.nl; Internet: http://www.telin.nl
Telephone: +31 (0)53-4850485; Fax: +31 (0)53-4850400

DISTRIBUTION OF MULTIMEDIA STREAMS TO MOBILE INTERNET USERS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. W.H.M. Zijm,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 20 mei 2005 om 15.00 uur

door

Cristian Engelbertus Wilhelmus Hesselman
geboren op 5 september 1970
te Hilversum

Dit proefschrift is goedgekeurd door: prof.dr.ir. E. Huizer (promotor), dr.ir. E.H. Eertink (assistent-promotor) en dr.ir. I.A. Widya (assistent-promotor).

Abstract

In this thesis, we consider the efficient distribution of live and scheduled multimedia content (e.g., radio and TV broadcasts) to mobile users via a ubiquitous wireless Internet. The objective is to design and develop a content delivery system that (1) enables content owners to deliver their multimedia content to a large number of heterogeneous receivers, and (2) enables receivers to continuously receive that content, independent of their location and the network they connect to.

Previous investigations into this topic have shown that multimedia content can be efficiently distributed through an overlay network that consists of multiple distributed proxy servers. In this thesis, we extend this concept to the distribution of live and scheduled multimedia content through multiple *aggregators*. An aggregator is an intermediary content provider that aggregates live multimedia content from various content sources (e.g., news services) and delivers it to mobile users through a pool of proxy servers. The availability of the same content through multiple aggregators enables mobile users to switch from one aggregator to another, thus alternately receiving the same content from different aggregators. The service area of an aggregator may be restricted to a certain set of networks, in which case switching to such an aggregator also requires mobile hosts to handoff to a network that is part of the aggregator's service area.

We call the system that switches a mobile host to another aggregator the ALIVE system, which stands for Aggregator Switching System for Mobile Receivers of Live Multimedia Streams. We concentrate on the 'front-end' of the ALIVE system, which supports mobile users and mobile hosts, aggregators, and wireless networks. We particularly focus on the signaling interactions between mobile hosts and aggregators and do not consider the details of the multimedia content itself.

The design of the ALIVE system is based on the *ALIVE business network*, which is a network of business roles (consisting of roles such as 'aggregator' and 'end-user') that describes the relations that may exist between domains

involved in the distribution of live multimedia content through multiple aggregators. In line with current trends in content distribution, the ALIVE business network consists of an application-level part (an overlay that consists of content sources and aggregators) and a network-level part (consists of providers of basic Internet access). We capture the properties of the relations in the business network in so-called "agreements".

The ALIVE business network uses the notion of a channel to refer to a particular piece of content (e.g., a TV broadcast). To further increase the number of potential receivers, aggregators in the ALIVE business network are able to transmit channels in various configurations. A configuration delivers a channel in a specific perceptual quality and requires a well-defined level of resources (e.g., network bandwidth). Aggregators may choose to support a relatively small number of configurations, thus striking a balance between per-user personalization of a channel (e.g., by delivering a channel in a configuration that is tailored to the instantaneous bandwidth available to a specific user's mobile host) and no personalization at all (i.e., everybody receiving a channel in the same configuration).

To support roaming users, aggregators establish application-level roaming agreements between each other. These agreements enable users to receive channels from multiple aggregators (e.g., at different locations) while having a subscription with only a few of them (typically one). Application-level roaming agreements define the configurations in which a user can receive channels from an aggregator with which the user does not have a subscription (called a foreign aggregator).

An aggregator may be bound to a specific set of networks through a so-called "binding agreement" with Internet access providers. In the ALIVE business network, such an aggregator is called a local aggregator because the binding agreement restricts its service area to the networks of the involved access provider. An Internet access provider may use local aggregators to offer exclusive channels or channel configurations to the users that connect to the access provider's networks (cf. the walled-garden models that contemporary cellular operators typically use).

The *ALIVE system* itself enables mobile users to roam in an unrestricted manner while continuously receiving a channel. The system transparently switches mobile hosts from one aggregator to another and executes handoffs on the mobile host's network interfaces. The system switches a mobile host to the aggregator that provides a certain channel in the best configuration, where 'best' is defined by the preferences of the end-user. This makes the ALIVE system a user-centric system.

The ALIVE system is scalable because most of its logic resides on mobile hosts (mobile-controlled switching). The system's operation is policy-controlled, which enables stakeholders in the ALIVE business network to

flexibly change the rules that the ALIVE system uses to make decisions (e.g., when to look for another available configuration of a channel).

The ALIVE system contains an application-level protocol, which we realized using the Session Initiation Protocol (SIP) and the Session Description Protocol (SDP), both of which are Internet standards. We deployed our implementation in a small-scale testbed with different types of networks, which represent the 'beyond 3G' Internet environment in which the ALIVE protocol is intended to operate.

Trough an *analysis* of our SIP-based implementation of the ALIVE protocol, we obtained quantitative information on how to smoothly execute switches between aggregators. Our analysis concentrates on the extra delay introduced by the ALIVE protocol in a contemporary wireless Internet environment, specifically consisting of 802.11 hotspots and UMTS/GPRS overlays. We focus on the operation of the ALIVE protocol immediately after a handoff to another 802.11 access provider, which is where the ALIVE protocol typically comes into play. After a handoff, the ALIVE system usually first attempts to discover the configurations in which the user can receive a channel from the local aggregators on the new network. At the edges of 802.11 cells, this may result is a significant delay because of the exponential back-off mechanism that SIP uses to recover from packet loss.

Our analysis consists of two parts: (1) a heuristic analysis of the application and network-level delay components involved in a typical switch and an estimation of their best-case values, and (2) an empirical analysis of the delay introduced by SIP transactions under various 802.11 network conditions. The analysis shows that the ALIVE system usually experiences little delay, except at the very edge of an 802.11 cell.

Based on our implementation and measurement work, we conclude that the ALIVE system is a feasible system that provides a clear contribution to the multimedia-everywhere vision.

Acknowledgements

This thesis is the result of a little over five years of work. During that time, I worked with quite a number of people and enjoyed their ideas, feedback, and support. I'd like to mention a few of these people here.

Erik Huizer became my professor in early 2001 when I only knew that I wanted to do 'something' with distributing multimedia streams in a wireless/mobile Internet environment. Erik helped me to shape those ideas and to focus my research, thus steering the whole endeavor toward what it has become today.

Henk Eertink has been my primary advisor during the past five years. Henk's broad technical knowledge and continuous stream of ideas were essential to my work. His enthusiasm and optimism were essential to my motivation to complete it.

Ing Widya was my advisor when I did my Master's and continued this job during my Ph.D. studies. His high-quality feedback and his ability to look at things from different angles greatly improved the quality of my work as well as of this thesis.

Hans Zandbelt, Arjan Peddemors, and Remco Poortinga helped out with the implementation of the system I built as well as with the many problems that popped up during the experiments with the system. Roy Arends provided the necessary support to configure the Telematica Instituut's lab network such that I could conduct my experiments.

Malohat Kamilova worked on the policy part of my research. Her questions and comments forced me to reflect on the work I had done when she started in late 2003 and clearly helped to improve and refine my work.

Erwin Fielt and Andrew Tokmakoff provided valuable feedback and ideas, in particular during our weekly Friday nights in downtown Enschede.

Other people who supported me are Mortaza Bargh, Geert Heijenk, Maarten Wegdam, Mark van Setten, Johan de Heer, and Margit Biemans.

I also thank my parents Jo and Engelbert Hesselman for their encouragements and interest. Final thanks go to my girlfriend Marloes Kroeze for her support and for putting up with my techno talk.

Contents

Chapter 1

Introduction

The landscape of wireless and mobile networks has changed considerably during the last decade. This holds for local personal area networks (like Bluetooth or UWB), nomadic networks (for instance wireless LANs) and mobile networks (such as GSM and UMTS). *Figure 1-1* illustrates this (taken from [MobileIT04]).

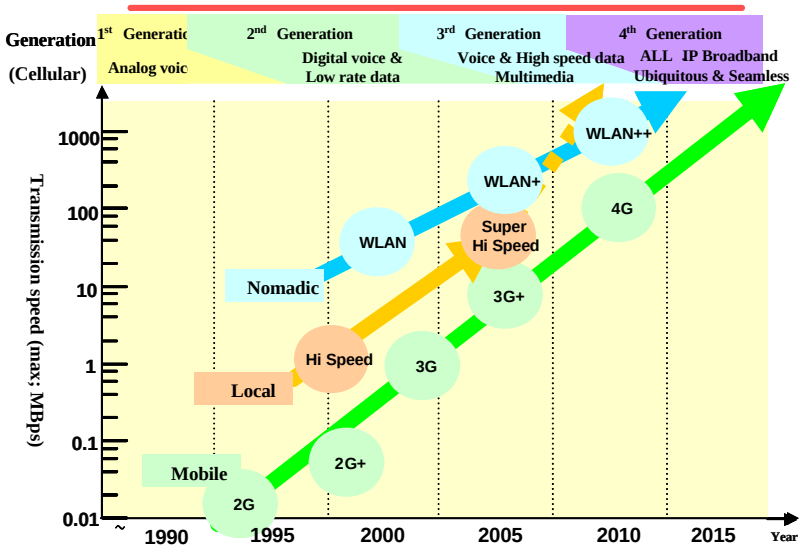


Figure 1-1. Developments in wireless and mobile networks.

An important observation is that the advances in networking *increase* the number of available networking technologies. For example, GSM and GPRS technologies will not disappear once UMTS is rolled out, and neither will 802.11 networks when their 100 MBit/s successors become available.

At the same time, the capabilities of new generations of mobile devices continue to increase as well [Rasmusson04]. There is a strong convergence between mobile phones and computers, resulting in smart phones with

multimedia capabilities, and personal digital assistants with telephony capabilities. Also, in the home the consumer electronics equipment and the computer integrate. The prime example of this is the launch in 2004 of Windows Media Center edition, a direct competitor of advanced consumer electronics devices and TV sets.

These developments lead to a large number of new distribution channels for providers of multimedia content. TV and radio channels, for instance, can be broadcast over any Internet access technology instead of via broadcast networks only.

This thesis must be read in the light of these trends and developments. It focuses on the delivery of multimedia live and scheduled content to mobile users.

1.1 Challenges and Objectives

In this thesis, we concentrate on the delivery of live and scheduled multimedia content (e.g., radio or TV broadcasts) over a ubiquitous Internet infrastructure consisting of different types of (wireless) access networks. Since the capabilities of networks and mobile hosts typically vary [Haardt00, Drew01], our first challenge is:

How can content providers deliver live multimedia content to a potentially large number of mobile hosts that have varying capabilities and connect to the Internet through different types of (wireless) networks?

One way to accomplish this is through an overlay network, which is usually referred to as a content distribution network [Wee03, Plagemann03, Day01]. For content distribution to mobile hosts, a few specific approaches exist in which multimedia content is distributed through multiple distributed proxy servers, with mobile hosts switching from one server to another as a result of mobility (e.g., because different proxy servers serve different networks) [Dutta02, Kim01, Roy02, Trossen03]. In this thesis, we extend this concept to the distribution of live and scheduled multimedia content through multiple *aggregators*. An aggregator is an access-controlled intermediary service provider that aggregates live multimedia content from content sources (e.g., cnn.com[1]) and delivers it to mobile users through a pool of proxy servers. The availability of a the same content through multiple aggregators enables mobile users to *switch* from one aggregator to another, thus alternately receiving the same content from different aggregators. The service area of an aggregator may be restricted to a certain

[1] The domain names in this thesis are for illustrative purposes only.

set of networks, in which case a switch to such an aggregator also requires a mobile host to *handoff* to a network that is part of the target aggregator's service area.

From the perspective of end-users, it is important that they continue to receive a particular multimedia transmission despite switches between aggregators and handoffs between networks. This is our second major challenge:

How can mobile users seamlessly receive a multimedia transmission in a way they consider best while they roam across different aggregators or access networks?

The *objective* of this thesis is to design a system that meets the challenges outlined above. We call this system the **ALIVE** system, which stands for **A**ggregator Switching System for Mobile Receivers of **Live** Multimedia Streams.

Our work concentrates on the 'front-end' of the distribution network, which consists of mobile users and mobile hosts, aggregators, and wireless networks. We furthermore focus on the signaling interactions between mobile hosts and aggregators and do not consider the specifics of the multimedia content itself (e.g., in terms of packet forwarding, compression, and packetization mechanisms).

1.2 Approach and Structure

We follow an approach that consists of three steps: the design of the ALIVE business network (step 1), the design of the ALIVE system (step 2), and an analysis of the ALIVE system (step 3). The ALIVE business network forms the foundation of the ALIVE system.

Step 1: Design of the ALIVE Business Network
In step 1, we design the ALIVE business network. The ALIVE business network is a network of business roles (e.g., consisting of roles such as 'aggregator' and 'end-user') that describes the relations that can exist between domains involved in the distribution of live multimedia content to mobile Internet users.

The key characteristic of the ALIVE business network is that it allows content sources to distribute the same multimedia content via multiple aggregators, which enables mobile users to receive that content from any of these aggregators. The business network follows current trends in content distribution in that it consists of an application-level part (with content sources and aggregators) and a network-level part (with providers of Internet access).

To maximize the number of potential receivers in a heterogeneous environment, the aggregators in the ALIVE business network are able to transmit content in various configurations. Different configurations typically deliver the same content in different perceptual qualities and require different amounts of resources (e.g., network bandwidth). The configurations that an aggregator supports typically sample the 'configuration spectrum' in a course way, thus striking a balance between one-configuration-for-all and individual per-user configurations (e.g., fine-tuned to a user's instantaneously available bandwidth).

The ALIVE business network captures the properties of the relations in the business network in so-called agreements. The distinctive type of agreement in the ALIVE business network is that of an application-level roaming agreement, which is an agreement between two aggregators that enables users of one aggregator to receive content from the other aggregator without having an agreement (subscription) with that aggregator. An application-level roaming agreement also defines in which configurations a user can receive content from a foreign aggregator. Application-level roaming agreements are similar to traditional network-level agreements, except that they contain application-level information (mappings between configurations) instead of information about network connectivity.

We discuss the ALIVE business network in Chapter 2 of this thesis.

Step 2: Design of the ALIVE System

In step 2, we design the ALIVE system. The goal of the ALIVE system is to automatically switch mobile hosts to the best possible aggregator in 'mid-call', where the user's preferences define what constitutes 'best'. Automatic switching frees users from having to manually switch to another aggregator, which enables them to make use of a multiple-choice environment in a user-friendly manner [Kleinrock03, Latvakoski02, Satyanarayanan01]. This is particularly important in a mobile environment where the set of available aggregators can change as a result of user mobility (e.g., when a user roams into a network where he can use new aggregators that are unavailable outside that network).

The system design puts as much of the system's responsibilities with the mobile host as possible, which is in line with current Internet design principles and makes the system scalable. An alternative solution is to put the system's responsibilities in access routers [Trossen03]. The advantage of this approach is that it integrates with Mobile IP [Solomon98]; the downside is that it requires a much more advanced and complex router infrastructure.

The behavior of the ALIVE system is driven by policies, which enables stakeholders in the ALIVE system/business network (e.g., users and companies renting mobile hosts) to flexibly change the system's behavior.

One of the components in the ALIVE system is the ALIVE protocol, which is responsible for the interactions between mobile hosts and aggregators. The ALIVE protocol is an application-level protocol that uses a minimal number of interactions between mobile hosts and aggregators to facilitate smooth switching. We implemented the ALIVE protocol using standard Internet protocols, specifically the Session Initiation Protocol (SIP) and the Session Description Protocol (SDP).

We discuss the design of the ALIVE business network in Chapter 3.

Step 3: Analysis of the ALIVE System

In step 3, we analyze the delay incurred by our implementation of the ALIVE protocol (i.e., based on SIP). We focus on an environment with 802.11 hotspots, where the ALIVE protocol typically comes into play at the edge of an 802.11 cell. At these edges, we experiment with the delays introduced by the ALIVE protocol, which may be substantial as a result of the exponential back-off retransmission scheme used by SIP to recover from packet loss. Such delays may hinder smooth switching.

We discuss our analysis in Chapter 4.

Conclusions and Related Work

Chapter 5 provides conclusions and considers topics for future work. Related work can be found at the end of each chapter.

Chapter 2

The ALIVE Business Network[2]

This chapter considers the design of the ALIVE business network. The ALIVE business network is a network of business roles that describes the possible relations between domains involved in the distribution of live and scheduled multimedia content to mobile Internet users. The network revolves around the notion of an aggregator, which is a role that receives live multimedia content from content sources (e.g., cnn.com) and forwards it to mobile users. The distinctive characteristic of the ALIVE business network is that it allows the same content (e.g., 'CNN TV') to be simultaneously distributed via multiple aggregators. This enables users to receive a certain part of a live multimedia transmission from one aggregator at one point and then switch to another aggregator to continue to receive the transmission from the new aggregator. The ALIVE business network forms the foundation of the ALIVE system (Chapter 3), which automatically switches a user's mobile host to the best aggregator.

This chapter begins with a brief explanation of how we interpret the notion of a business network (Section 2.1). Next, we discuss the ALIVE business network itself, which consists of three parts: an application-level part that primarily deals with content distribution (Section 2.2), a network-level part that focuses on transporting IP packets (Section 2.3), and a cross-level part that ties these two levels together (Section 2.4). The emphasis of our work is on the application and cross-level parts. We conclude this chapter with a discussion on related work (Section 2.5) and a summary (Section 2.6).

2.1 Business Networks

In this thesis, we think of a business network as a graph in which the nodes are *business roles* (Section 2.1.1) and the edges are *business relations* (Section 2.1.2). Both of these concepts are inspired by the corresponding notions in the TINA Business Model [TINA97].

[2] This chapter is based on [Hesselman02], with updates from [Hesselman03] and [Hesselman05].

2.1.1 Business Roles

The delivery of a (multimedia) service to end-users generally requires a number of high-level activities such as 'content generation', 'data transport', and 'content play back'. A business role encompasses a subset of these activities, which may be assigned to the business role for economical, technical, or legislative reasons [TINA97]. Examples of business roles are content providers, IP connectivity providers, terminal providers [DOLMEN98], location owners, infrastructure owners [Verhoosel03], and so forth.

Business Networks

A business network consists of a number of business roles. Each role in the network is responsible for a certain part of the activities to deliver a service to end-users. In this thesis, we assume that end-users are persons and not organizations.

Figure 2-1 shows an example of a simple business network in which the role 'content provider' is responsible for transmitting live multimedia content to Internet users (1), and the role 'access provider' is responsible for transporting content to and from the Internet backbone in the form of IP packets (2, 3).

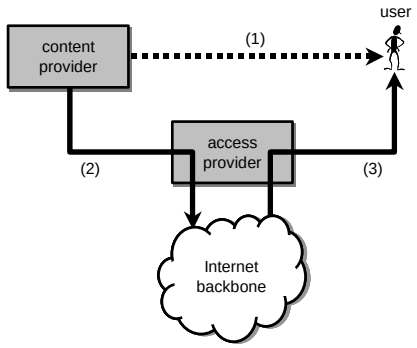


Figure 2-1. Simple example of a business network.

Figure 2-1 also shows that the grouping of activities into business roles can be driven by different criteria. For example, the motivation to distinguish content providers on the one hand and access providers on the other is primarily a technical one (processing multimedia content is quit different from transporting it). At the same time, the distinction is also economical: it is very likely that content providers are willing to pay for the transport services of access providers to increase the number of viewers of their content.

Administrative Domains

An *administrative domain* is a set of computing and communications devices (e.g., servers, routers, base stations, and so on) owned by a single organization or person. An administrative domain plays one or more roles and uses its devices to realize the activities associated with these roles. For example, content providers use servers that enable clients to retrieve (multimedia) content, and access and backbone providers use (sub) IP-level devices (e.g., routers and base stations) to transport IP traffic.

Figure 2-2 shows an example of how the roles of the business network of *Figure 2-1* can be distributed across administrative domains, in this case broadcast.com, access.com, and hotspot.nl.

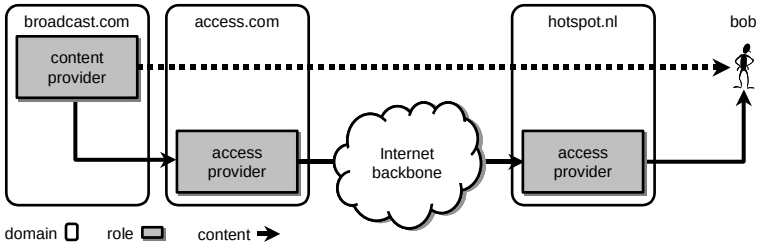


Figure 2-2. Distribution of roles across domains.

In general, each domain can play one or more roles. For example, domain hotspot.nl could also be a content provider, for instance by inserting local ads into the multimedia content coming from broadcast.com (cf. [Dutta02]). The examples in this thesis will however mostly involve domains playing one role.

For simplicity, we only use domain names in this thesis. We do not distinguish between a domain name (e.g., broadcast.com) and the owner of that domain (which may be company X that also owns access.com).

2.1.2 Business Relationships

Business roles are connected into a business network by business relationships. A business relationship is a *user-provider relationship* [Halteren99, TINA97], which can for instance be of an economical nature (end-users paying providers for their services) or of a technical nature (e.g., content providers making use of access providers) [TINA97]. The example business network of *Figure 2-1* involves three user-provider relationships: one between the content provider and the end-user, one between the content provider and the access provider (the content provider being the user), and one between the end-user and the access provider.

A business relationship may be a bi-directional user-provider relationship (e.g., a roaming relationship between two GSM access providers), which is usually referred to as a federation [DOLMEN98].

Agreements

We describe the properties of a business relationship in an *agreement*. An agreement typically specifies which services the user role in the relationship can receive from the provider role (e.g., at most 1 Mbps of best-effort downstream connectivity). This may also include a pricing model and user and provider obligations (e.g., the amount of up time), but these two topics are outside the scope of this thesis.

In general, agreements can be established in many ways. They can for instance be set up in an on-line or an off-line manner, or bilaterally or through a broker [3GPP99]. In addition, agreements can exist for a 'longer' or a 'shorter' period of time (e.g., as long as a user receives a service from a provider) [DOLMEN98]. In this thesis, we do not make any assumptions on the way in which agreements are established or on how long they exist. We abstract away from these issues by assuming that the agreements that our model requires (see sections 2.2, 2.3, and 2.4) have already been established.

2.2 Application-level Part: Content Distribution

Similar to other business networks (e.g., [TINA97, DOLMEN98, Vernick01]), the ALIVE network consists of an application-level part and a network-level part. This section discusses the application-level part of the ALIVE business network, which consists of business roles that are responsible for delivering live or scheduled multimedia content (e.g., a TV broadcast) to mobile users over the Internet. The central notion is that of a *content aggregator*, which is a business role that redistributes live multimedia content (e.g., a TV broadcast) to mobile users.

We first discuss the application-level roles of the business network (Section 2.2.1) and the relations that exist between them (Section 2.2.2). After that, we briefly consider alternative types of business networks (Section 2.2.3) and take a look at the notion of a configuration (Section 2.2.4), which is set of streams that carry a piece of multimedia content using specific compression and packetization parameters. Next, we discuss the application-level agreements of the ALIVE business network (Section 2.2.5).

2.2.1 Sources and Aggregators

The application-level roles of the ALIVE business network are those of a content source and a content aggregator. Together, they deliver *content channels* to mobile users. In this thesis, a content channel is the logical content that end-users receive (e.g., 'CNN TV').

Content Source
A content source (e.g., cnn.com) is the origin of one or more channels. In this thesis, we assume that each channel originates at one source. A source consists of one or more media servers.

Content Aggregator
The central role in the ALIVE business network is that of a content aggregator. A content aggregator receives channels from sources and redistributes them to mobile users. In general, it is possible to deliver channels to mobile users through an aggregator hierarchy of depth $h \geq 1$ (cf. the clusters of [Chawathe02]). To keep our business network simple, we assume that h equals one in this thesis.

An aggregator typically transmits channels off a pool of media servers (cf. [Chawathe02, Amir98]).

Users and Mobile Hosts
We assume that each user is logged onto one mobile host and that a mobile host is used by one user. As a result, we use the terms mobile user and mobile host interchangeably, unless the distinction is required.

2.2.2 Business Network

The ALIVE business network is organized such that (1) sources distribute channels through aggregators and (2) users need to set up an agreement with aggregators to be able to receive channels. We call this a *portal* type of network since aggregators shield sources off from users. We refer to Section 2.2.3 for a discussion on alternative business network types (e.g., a pure end-to-end model) and to Section 2.2.5 for the details on the contents of the agreements between users and aggregators.

Multi-Aggregator Distribution
The distinctive characteristic of the ALIVE business network is that it allows the same channel (e.g., CNN TV) to be simultaneously distributed via *multiple* aggregators. This enables users to receive that content from one aggregator at one point and then make a switch to receive the content from another aggregator (e.g., when the current aggregator becomes unavailable as a result of the user moving into another network). Application-level roaming agreements between aggregators facilitate these switches in that they enable users to receive content from multiple aggregators while having a subscription (agreement) with only a few of them (typically one). We will discuss the contents of these roaming agreements in Section 2.2.5.

Figure 2-3 shows a typical instance of the ALIVE business network in which user Bob can receive channels CNN TV and BBC Radio through two

aggregators, stream-it.com and media-forward.nl. In this specific example, Bob has an agreement with media-forward.nl, but receives CNN TV from stream-it.com. This is possible because the two aggregators have set up an application-level roaming agreement. *Figure 2-3* shows that sources and aggregators need to set up agreements as well. In the ALIVE business network, these agreements define how aggregators can forward channels (e.g., if they are allowed to transcode them to a lower quality). We will discuss them in more detail in Section 2.2.5. Observe that *Figure 2-3* does not show Bob's mobile host.

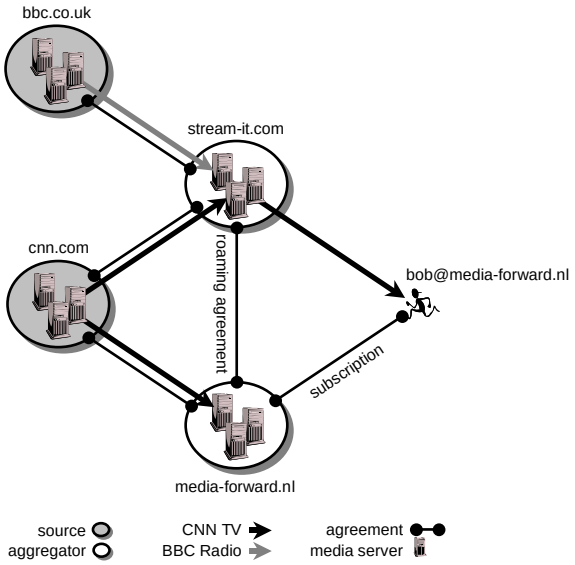


Figure 2-3. Example use of sources and aggregators.

Pros and Cons for Sources

A source typically benefits from a portal type of business network because aggregators increase the source's scalability. This is particularly important when the source has to serve a potentially large number of users, as is the case in the ALIVE business network. Instead of unicasting a channel N times to N mobile hosts, a source could for instance unicast them to a small number of aggregators. Each of the aggregators would then forward the channels to a subset of the N mobile hosts. This scheme reduces the load on the source (it has fewer concurrent outstanding connections to deal with) and will also reduce its bandwidth consumption because it does not have to transmit N copies of the same channel. Both of these aspects increase the scalability of the source [Rosenberg98].

IP Multicast can also reduce the number of concurrent connections that a source has to deal with, for instance by multicasting a channel from a

source directly to receivers (e.g., [Wu97, McCanne96, Cheung96]), or by using IP Multicast on the paths where it is available (e.g., between the source and a number of aggregators). However, the usefulness of such schemes depends on the availability of IP Multicast, which is limited at this point [Chennikara02].

Reduced bandwidth consumption has the side effect that it typically also reduces the source's transit costs [Norton02a], which are the costs a source has to pay to its local Internet connectivity provider to get its traffic transported to the Internet. Transit costs usually depend on the number of megabits per second (Mbps) that a source transmits [Norton02b].

Sources also benefit from aggregators because they can off-load certain media processing tasks (e.g., transcoding) to them, which reduces the sources' resource requirements and thus increases their scalability [Gao03]. A proxy could for instance free a source from transcoding a channel in HDTV quality down to a quality suitable for mobile hosts. Media processing operations like transcoding have been studied extensively in the literature (e.g., [Xu00, Amir95, Yeadon96, Balachandran97, Zenel97, Roy02]), but specific media processing operations are outside the scope of this thesis.

Pros and Cons for Users

Users benefit from the existence of aggregators when a source simultaneously distributes a channel via *multiple* aggregators (cf. [Roy02, Dutta02, Trossen03]). In this case, users might be able to receive that channel from more than one aggregator, which enables them to choose the aggregator they consider the 'best' one (e.g., the one that delivers the channel at the lowest price). This is in line with broader economical trends that provide users with ever more options (e.g., multiple power and telco providers).

At the same time, the availability of multiple aggregators also requires users to actually deal with these aggregators, which is a task that might be difficult [Kleinrock03, Latvakosi02]. This problem may be exacerbated by roaming. For example, if a user is receiving a channel from an aggregator that is bound to a certain network, then that user will have to find another aggregator when he moves out of the network. A solution to this problem is a system that can automatically switch a mobile host to another aggregator while the user is receiving a channel. In Chapter 3, we will consider the design and implementation of the ALIVE system, which realizes such switches.

Another advantage for users is that aggregators typically offer multiple channels, which enables users to get access to these channels by establishing an agreement with only a small number of aggregators (typically one). Without aggregators, users would have to set up an agreement with every source from which they would want to receive a channel.

Aggregators from an Internet Perspective

There has been a lot of debate in the Internet engineering community (in particular in the IETF) about the use of intermediaries such as the aggregators we use in this thesis. The Internet Architecture Board (IAB) recently issued an RFC [Floyd02] in which they state that proxies are acceptable if their use is authorized by the content provider or the receiver of the content. This requirements holds in the ALIVE business network as users explicitly establish agreements with aggregators, which authorizes the use of that aggregator. In addition, content providers (our sources) also establish agreements with aggregators, which also authorizes the use of intermediaries (aggregators) from the content provider's perspective.

The IAB furthermore requires that intermediaries are explicitly addressed at the IP layer. As we will see in Chapter 3, this is indeed the case in the ALIVE system.

2.2.3 Alternative Business Networks

In general, users can set up an agreement with either sources or aggregators and can then also receive a channel from either a source or an aggregator. *Figure 2-4* illustrates that this yields four types of distribution models:

- An end-to-end model, in which users receive channels from sources and also have agreements with sources;
- A Content Distribution Network (CDN) model, in which a mobile user has an agreement with a source (e.g., cnn.com), but receives channels through an aggregator that the source has contracted (cf. akamai.com). In this form of distribution, a mobile host sends a request for a channel to a source. The source forwards the request to an aggregator (the CDN), which routes the request [Cain03] to a media server that is 'close' to the user (e.g., http://nearest-server.akamai.com/CO231234), possibly in a resource-aware manner [Xu00].
- A brokered model, in which a user sets up an agreement with an aggregator, but receives channels from a source. In this case, the aggregator basically acts as a broker (cf. the Broker role in TINA [TINA97]) that merely enables users to look up channels from contracted sources, for instance using an Electronic Program Guide (EPG) on the user's mobile host [Nomura03]; and
- A portal model, in which a user receives a channel from an aggregator and also has an agreement with that aggregator. This is the model that forms the basis of the ALIVE business network's organization (see Section 2.2.2).

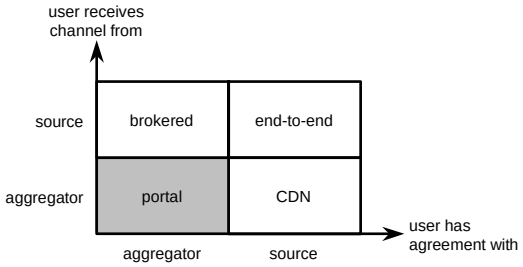


Figure 2-4. Distribution types.

Figure 2-5 shows an example of the end-to-end model (*Figure 2-5a*), the CDN model (*Figure 2-5b*), the portal model (*Figure 2-5c* and *Figure 2-3*), and the brokered model (*Figure 2-5c*).

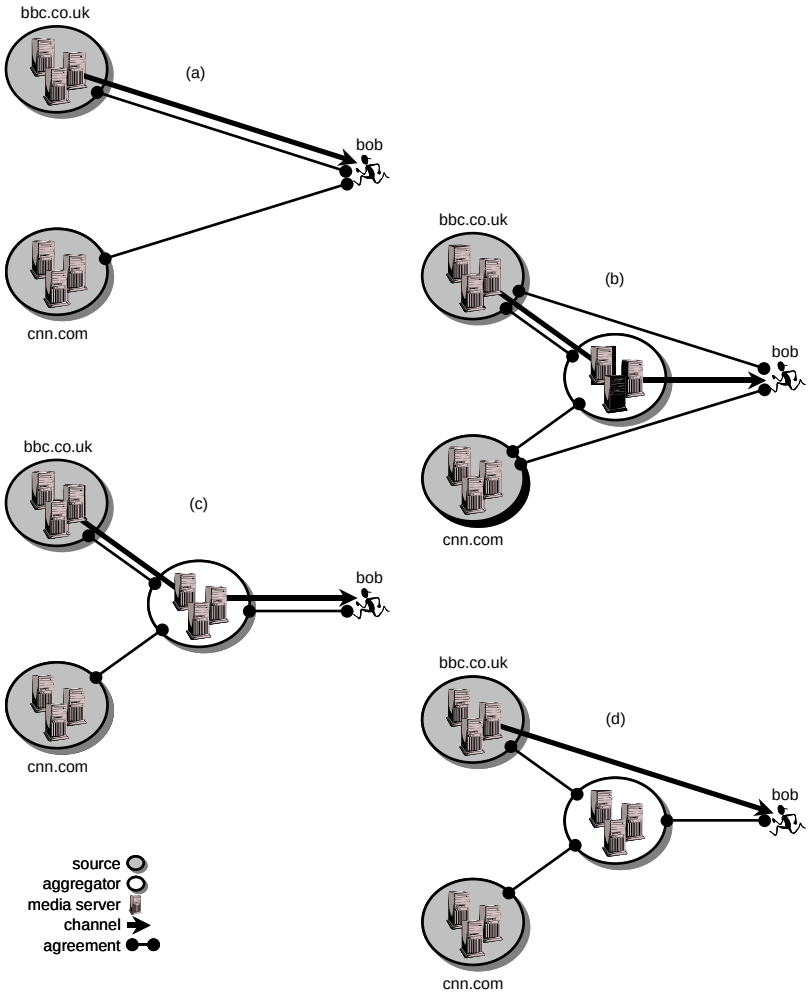


Figure 2-5. Examples of distribution types: end-to-end (a), CDN (b), portal (c), and brokered (d).

Observe that the three models that involve an aggregator can also describe the end-to-end model by co-locating a source and an aggregator (i.e., when a domain is both a source and an aggregator). This makes the end-to-end model a special case of the other three models.

2.2.4 Configurations

To maximize the number of potential receivers of a channel in a heterogeneous environment, the sources and aggregators in the ALIVE business network are able to transmit channels in various *configurations*. A configuration delivers a channel in a certain form, for instance in terms of perceptual quality (e.g., in terms of frame rate, pixels per frame, and colors per pixel), costs, and resource requirements (e.g., in terms of the processing capabilities that mobile hosts need to possess to receive a channel in a certain configuration).

A configuration consists of set of multimedia streams with specific compression and packetization parameters (e.g., in terms of a codec type, a compression ratio, a sampling rate, and a packetization format) [Xu00, Plagemann03] and is parameterized by the channel (i.e., the logical content) the streams carry. As a result, we speak of *a particular channel in a certain configuration*. Examples are CNN Radio in a 64 kbps MP3 audio configuration, CNN Radio in a 32 kbps G722.1 audio configuration, and so on. The packets of a multimedia stream are typically formatted according to one of the profiles of the Real-time Transport Protocol (RTP) [Schulzrinne96a, Schulzrinne96b].

Supported Configurations

Sources and aggregators each support their own configurations in which they can potentially deliver channels to receivers. We call these a source's (aggregator's) set of supported configurations:

A supported configuration is a configuration in which a source or an aggregator can potentially deliver a channel to receivers.

In the rest of this thesis, we assume that the supported configurations of aggregators are suitable for mobile hosts and wireless links ('mobile-friendly' configurations), while this *may* be the case for the supported configurations of sources. If a source's supported configurations are mobile friendly, then aggregators can simply reuse those configurations (i.e., the supported configurations of the source and aggregator overlap). Alternatively, aggregators could use a set of supported configurations that completely differs from those of a source, for instance because the source's configurations are unsuitable for the typical processing capabilities of

mobile hosts. Notice that mobile friendly configurations typically provide a lower perceptual quality level than those that are not.

As will see in Section 2.2.5, supported configurations form the foundation of the agreements in the ALIVE business network.

Personalization

We expect that the set of supported configurations of an aggregator (source) will be relatively small (e.g., in the order of 10 to 20), thus sampling the 'configuration spectrum' in a coarse way. The advantage of this approach is that it increases the scalability of aggregators (sources) because it limits the amount of per-user multimedia state (e.g., pointers to multimedia files, transcoders, and so on) they have to maintain. The downside is that users might receive a channel in a suboptimal configuration given their quality and price preferences and the capabilities of their mobile hosts. The network connection of a user could for instance provide some extra bandwidth, but not enough to receive a channel in the next higher configuration. The ALIVE business network thus strikes a *balance* between offering a channel in a single configuration for everyone and using (a large number of) configurations that are optimized for individual users and their mobile hosts (e.g., fine-tuned to their instantaneously available bandwidth).

The above personalization problem also surfaces in the distribution of multimedia channels through IP multicast. The problem here is that it is difficult to multicast a multimedia channel in an optimal way to a heterogeneous set of receivers. Some receivers may for instance experience packet drops because they are sitting behind a (congested) link that cannot handle the channel's bandwidth level. Other receivers may be able to receive the channel at a higher quality because they connect to the Internet through a high-capacity congestion-free link. This problem can be alleviated by using multiple multicast groups. Each multicast group could for instance carry one layer of a layered encoder and receivers would then dynamically add or drop layers by joining or leaving the appropriate multicast groups (e.g., [McCanne96, Wu97]). Alternatively, the multicast groups could form a hierarchy in which each multicast group distributes a channel at a specific bandwidth level [Kouvelas98], or sources could simulcast a channel at different bandwidth levels onto multiple multicast groups and then rely on receivers to switch between these groups [Cheung96].

Switching versus Adaptation

When a user switches to receive a channel from another aggregator, it will typically also receive the channel in another configuration. Since aggregators usually sample the configuration spectrum in a course way (see Personalization), the target configuration might provide a significantly

different perceptual quality. We therefore consider switching a *coarse-grained* form of adaptation.

Course-grained adaptation is complementary to *fine-grained adaptation* [Karrer01], which adapts the characteristics of an individual configuration. An aggregator can for instance reduce the bandwidth that a channel configuration requires by dropping video frames (e.g., [Yeadon96]).

In this thesis, we will leave fine-grained adaptation up to native streaming technologies such as WindowsMedia and Real [Li02] and do not consider it any further.

Configuration Categories

To market their configurations, aggregators can package their supported configurations into groups, for instance based on the perceptual quality they provide, on the network bandwidth they require, or on the amount of battery power they require from receivers. In this thesis, we consider groups of configurations that provide a similar perceptual quality. We refer to them as *quality categories*. Quality categories are typically ordered [Xu00], but the specifics of such orderings are outside the scope of this thesis.

Aggregators associate their categories with a user-oriented *quality label* (e.g., 'CD' quality audio or 'TV' quality video). These quality labels are used in agreements with users (see Section 2.2.5) and can also be used to provide feedback on the quality level of the configuration in which a user actually receives a channel. Quality labels can also be used in combination with a pricing model. Different aggregators typically use different quality labels.

Configuration Definition

In general, the properties of a configuration can be defined by a source, by an aggregator, or by a standardization body. Standardized configurations have the same properties (e.g., perceptual quality) across different sources and aggregators. The grouping of configurations and their ordering could also be subject of standardization. The way in which configurations are defined is however outside the scope of this thesis.

2.2.5 Delivery, Roaming, and Forwarding Agreements

As we have seen in Section 2.2.2, the distribution of channels to mobile hosts is governed by three types of agreements: agreements between users and aggregators, agreements between aggregators, and agreements between sources and aggregators. We refer to these three types of agreements as delivery agreements, (application-level) roaming agreements, and forwarding agreements, respectively. Each agreement typically comes with a

pricing model, but accounting is a topic that lies outside the scope of this thesis.

Delivery and Forwarding Agreements

Delivery and forwarding agreements contain descriptions of *allowed configurations*, which are supported configurations (see Section 2.2.4) in which a receiver (i.e., an aggregator or a user) is allowed to receive channels from a sender (i.e., a source or an aggregator). That is,

An allowed configuration is a configuration in which a receiver (i.e., an aggregator or a user) can receive channels from a sender (i.e., a source or an aggregator). An allowed configuration must be a supported configuration of the sender and the allowed configuration's description must appear in the agreement between the sender and the receiver.

The actual description of an allowed configuration may for instance be in the form of its properties (e.g., its codec type, number of streams, and required bandwidth) or in the form of a label that identifies the quality category to which the configuration belongs (see Section 2.2.4).

Figure 2-6 (part of *Figure 2-3*) shows an example of a delivery agreement between user Bob and aggregator media-forward.nl, as well as of delivery agreements between cnn.com and the two aggregators (media-forward.nl and stream-it.com). As we will see in Chapter 3, aggregators are responsible for controlling access to their configurations, which involves the authentication of users and mapping their identities to a set of allowed configurations (authorization).

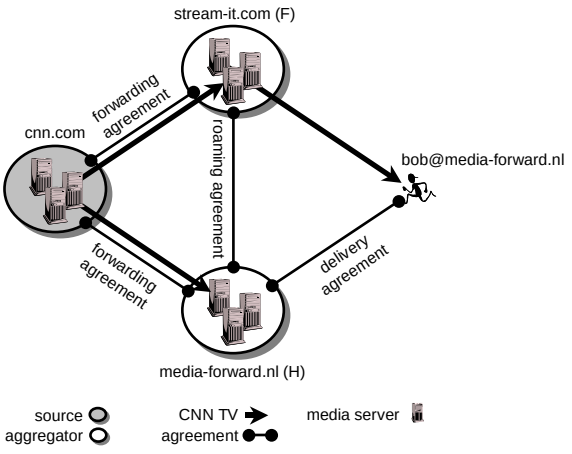


Figure 2-6. Multi-aggregator distribution, including agreements.

The difference between a forwarding agreement and a delivery agreement is that a forwarding agreement also specifies how an aggregator should handle

a source's channels. For example, a forwarding agreement could indicate if a source permits an aggregator to redistribute a channel in a configuration that differs from the one in which the aggregator receives the channel (e.g., by transcoding the original configuration to a low-bandwidth format), for instance using the MPEG-21 Rights Expression Language [Wang04]. In the rest of this thesis, we will however concentrate on the 'front-end' of the ALIVE business network (aggregators and users), which means that forwarding agreements are out of scope.

Home and Foreign Aggregators

From a user's perspective, we distinguish home and foreign aggregators. A *home aggregator* is an aggregator with which a user has a delivery agreement, whereas a *foreign aggregator* is an aggregator with which a user does not have such an agreement. In the example of *Figure 2-6*, media-forward.nl is Bob's home aggregator (marked with an 'H'), while stream-it.com is a foreign aggregator (marked with an 'F').

From an aggregator's perspective we also distinguish *foreign users*, which are users that do not have a delivery agreement with the aggregator.

Application-level Roaming Agreements

The home aggregators in the ALIVE business network establish *application-level roaming agreements* with foreign aggregators to enable their users to also receive channels via those foreign aggregators. For example, Bob's home aggregator media-forward.nl has a roaming agreement with foreign aggregator stream-it.com (*Figure 2-6*) so that media-forward.nl's users can also receive channels from stream-it.com.

An application-level roaming agreement defines an equivalence relationship between the supported configurations of the home aggregator and those of the foreign aggregator. The equivalence relation may be based on factors such as bandwidth, cost, or perceptual quality. *Figure 2-7* shows an example of an equivalence relation between the audio configurations of aggregators media-forward.nl and stream-it.com. It for instance specifies that 24 kbps G722.1 configuration of stream-it.com is equivalent to either of the two G722.1 configurations of media-forward.nl (e.g., because they provide about the same quality).

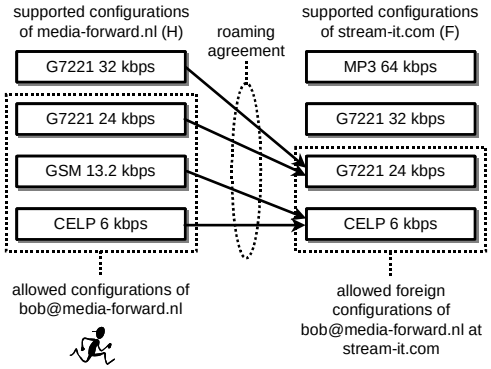


Figure 2-7. Example of a roaming agreement.

A delivery agreement and a roaming agreement together determine the set of configurations in which a user is allowed to receive configurations from a foreign aggregator. We therefore refer to this set as the user's *allowed foreign configurations*.

In the example of *Figure 2-7*, Bob's delivery agreement with media-forward.nl (his home aggregator) contains three allowed configurations. The roaming agreement with stream-it.com reduces this set to two configurations at stream-it.com, which are Bob's allowed foreign configurations at that aggregator.

Application-level roaming agreements are similar to *network-level roaming agreements* (e.g., [Markoulidakis97, 3GPP99]), but they only address application-level issues (i.e., equivalent configurations). Like a network-level roaming agreement, an application-level roaming agreement does *not* include user-specific information.

Multiple Sets of Allowed Configurations

In general, the total set of allowed configurations in which a user can receive a channel may involve allowed configurations of different aggregators. For example, the total set of allowed configuration in which Bob can receive CNN TV (*Figure 2-6*) consists of allowed configurations of media-forward.nl and of the allowed foreign configurations of stream-it.com.

To actually deliver a channel to the user, it should somehow be possible to select a configuration from these different sets of allowed configurations in which the user will actually receive the channel. *Figure 2-8* illustrates this.

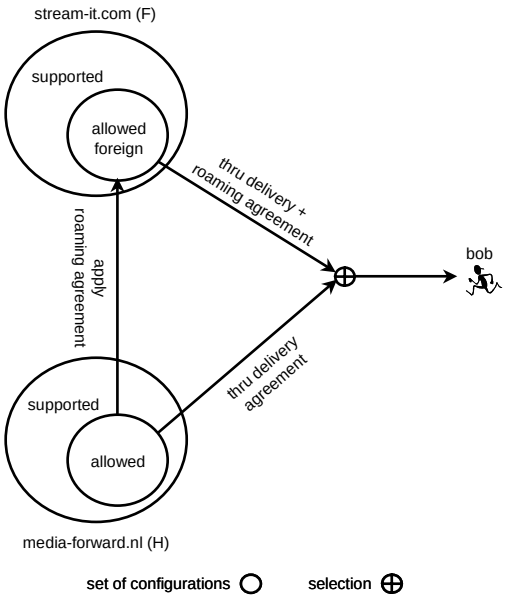


Figure 2-8. Configuration selection using multiple sets of allowed configurations at different aggregators.

As we will see in Chapter 3, the ALIVE system can automatically select an actual configuration for a particular user.

2.3 Network-level Part: IP Connectivity

The roles in the network-level part of the ALIVE business network provide end-to-end IP connectivity to sources, aggregators, and users. Their primary task is to transport IP packets from one Internet host to another.

In this section, we first discuss the network-level roles of the ALIVE business network (Section 2.3.1) and take a look at how they are organized (Section 2.3.2). After that, we discuss the network-level agreements (Section 2.3.3).

2.3.1 Access Providers and Backbone Providers

At the network-level, we distinguish two roles: access providers and backbone providers.

Access Provider

An access provider operates at the fringes of the Internet and provides first-hop IP connectivity to users, aggregators, and sources (cf. the model of [Rosenberg98]) at various bandwidth levels. Mobile users usually receive a channel via one access provider, which is typically a wireless one. In general,

a single access provider can operate different types of (wireless) access networks (e.g., 802.11, 802.16, or UMTS).

To limit the complexity of the ALIVE business network, we only require access providers to deliver the standard best-effort packet delivery service. We do not require them to possess special features like the ability to provide Quality of Service (QoS) assurances [Xiao99] or to be able to handle mobility (e.g., using Mobile IP [Solomon98]).

Backbone Provider

A backbone provider offers IP-level connectivity to access providers. Backbone providers make up the Internet backbone and do not serve mobile users, sources, or aggregators.

Scope

Since the focus of our work is on the front-end of the ALIVE business network, we will not consider individual backbone providers in this thesis. Instead, we will group them together in one 'Internet backbone' cloud.

2.3.2 Business Network

At the network-level, the ALIVE business network consists of access providers that receive IP packets from the Internet backbone and deliver them the hosts of mobile users, or vice versa. Users need to set up an agreement with access providers to gain Internet access through one or more of those providers' networks.

Multiple Networks

In general, mobile users may be able to connect to the Internet through multiple networks of multiple access providers. To remain connected to the Internet, some sort of handoff system needs to transfer mobile hosts to another network when they leave the coverage area of their current network (e.g., [Solomon98, Wedlund99, Seneviratne98, Pollini96, Pahlavan00, Tripathi98]). Roaming agreements between access providers facilitate these handoffs in that they enable users to make use of networks of different access providers while having a subscription (an agreement) with only a small number of access providers [Verhoosel03, Markoulidakis97, 3GPP99] (typically one). We will consider these network-level roaming agreements in more detail in Section 2.3.3.

At certain locations, mobile hosts may be able to connect to multiple networks *simultaneously*. This typically happens when a mobile user resides in the coverage of two different types of networks (e.g., an 802.11 and a UMTS network). Such networks usually have an *overlay* relationship with each other, which means that one network also covers the coverage area of

the other [Stemm98, Brewer98]. To date, overlays typically involve a UMTS network (the overlaying network) and an 802.11 'hotspot' (the overlayed network) [Køien03, Banerjee04, Zhuang03].

Figure 2-9 shows an example of an overlay situation combined with the example aggregator infrastructure of *Figure 2-6*. In this specific example, Bob's mobile host can simultaneously connect to the 802.11 network of hotspot.nl and to the UMTS network of connect-it.nl as long as Bob is within the range of the 802.11 network. *Figure 2-9* does not show the access providers that the sources and the aggregators use to connect to the Internet, nor does it show the interconnecting backbone providers.

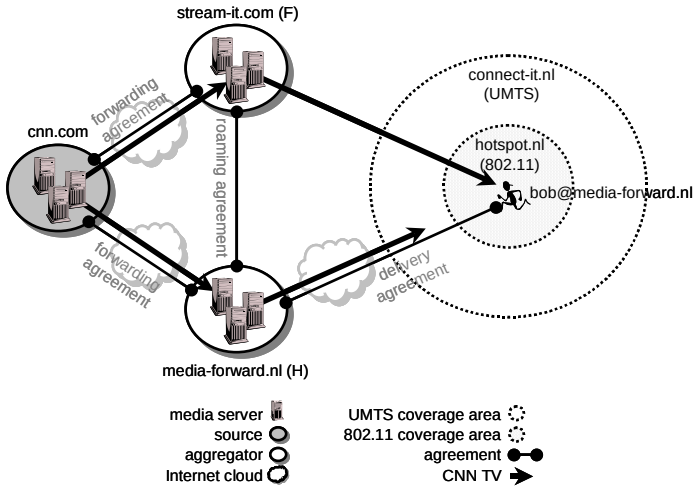


Figure 2-9. Instance of the ALIVE business network with access providers and backbone providers.

Multiple Aggregators, Multiple Networks

The availability of multiple networks not only enables users to receive a channel via multiple alternative aggregators, but also through multiple alternative networks. As we will see in Chapter 3, the ALIVE system is able to automatically switch between aggregators and handoff between networks.

2.3.3 Connectivity and Roaming Agreements

The agreements at the network-level are largely similar to those at the application-level, except that they contain network-level information. In this section, we only consider agreements between users and access providers and agreements between access providers. We refer to them as connectivity agreements and (network-level) roaming agreements, respectively. The agreements between sources and access providers, between aggregators and access providers, and between access providers

and backbone providers are outside the scope of this thesis. We also do not consider the pricing models that can be associated with the agreements.

Connectivity Agreements

A *connectivity agreement* between a user and an access provider allows the user to send/receive IP packets to/from the access provider. It specifies which of an access provider's networks a user can access and which bandwidth levels (upstream and downstream) that are available to that user on each of these networks. We refer to the networks that appear in a connectivity agreement as the user's allowed networks for that specific access provider (e.g., Bob could have a connectivity agreement with connect-it.nl that specifies that the UMTS network is an allowed network). Similarly, we call the bandwidth levels in such an agreement the user's allowed bandwidth levels.

The set of allowed networks is a subset of an access provider's set of supported networks, while the set of allowed bandwidth levels is a subset of an aggregator's set of supported bandwidth levels. (Notice the similarity with supported and allowed configurations at the application level.) The set of supported bandwidth levels could for instance contain up to 31 multiples of 64 kbps (cf. traditional ISDN networks), while the maximum allowed bandwidth level for a particular user is only 128 kbps.

Home and Foreign Access Providers

A user typically establishes a connectivity agreement with one access provider. We refer to this access providers as the user's *home access provider*. All other access providers are *foreign access providers*.

Roaming Agreements

A home access provider establishes *roaming agreements* with foreign access providers to give its users access to the networks of the foreign access providers. For example, if connect-it.nl is Bob's home access provider, then a roaming agreement between connect-you.nl and hotspot.nl would also give Bob access to hotspot.nl's 802.11 network.

A roaming agreement defines a mapping between the supported networks and supported bandwidth levels of two access providers. For example, the roaming agreement between connect-you.nl and hotspot.nl could specify that the 64/16 kbps (downstream/upstream) supported bandwidth level of media-forward.nl maps to the 512/64 kbps supported bandwidth level of hotspot.nl.

The allowed bandwidth levels that a user can receive from a foreign aggregator depends on the user's connectivity agreement and the roaming agreement between the foreign access provider and his home access provider. (Notice the similarity with the delivery and roaming agreements at the application level.)

2.4 Cross-Level Part: Scoped Content Distribution

The cross-level part of the ALIVE business network defines an optional binding between the application-level and network-level parts. This binding is established by means of agreements between aggregators and access providers.

We first discuss the roles associated with the cross-level part (Section 2.4.1) and then discuss the binding agreements (Section 2.4.2).

2.4.1 Local and Global Aggregators

The cross-level part specializes aggregators into global and local aggregators. A *local aggregator* is an aggregator whose service area is restricted to a number of networks. This means that a user can only receive channels from a local aggregator (in certain allowed configurations) if his mobile host attaches to one of the networks that belong to the aggregator's service area. A user has to switch to another aggregator if he receives a channel from a local aggregator and leaves that aggregator's service area. In general, the service area of a local aggregator may involve networks of different access providers.

Contrary to a local aggregator, a *global aggregator* is available in the entire Internet.

Figure 2-10 (an extension of the example of *Figure 2-9*) shows stream-it.com as a local aggregator (marked with an 'L'). Its service area is restricted to the 802.11 network of hotspot.nl. Media-forward.nl is a global aggregator (marked with a 'G') and is therefore available in the 802.11 network as well as in the UMTS network.

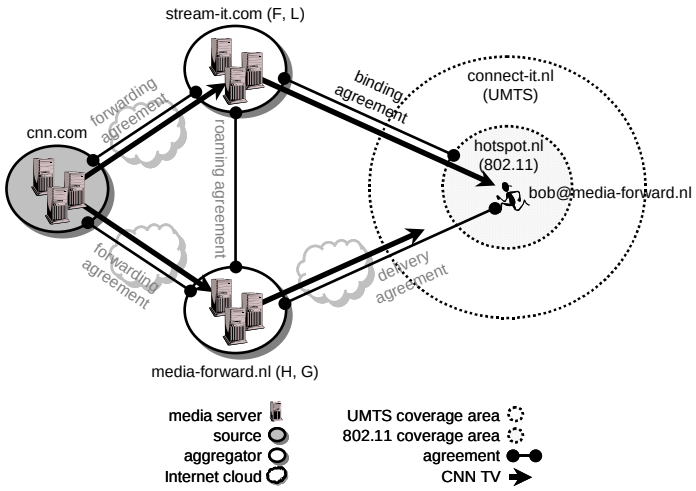


Figure 2-10. Example of global and local aggregators.

2.4.2 Binding Agreements

An aggregator and an access provider set up *a binding agreement* to define for which of the access provider's networks the aggregator acts as a local aggregator. As a result of the agreement, the involved aggregator's channels and configurations are only accessible through the networks listed in the agreement. The agreement between stream-it.com and hotspot.nl (*Figure 2-10*) is an example of a binding agreement. It lists in which networks of hotspot.nl stream-it.com acts as a local aggregator. In the example, this is hotspot.nl's 802.11 network.

An example of a practical situation in which binding agreements were used is in the distribution of live broadcasts of the 2004 Olympic Games over the Internet [Wired04]. In this particular case, the International Olympic Committee (IOC) acted as the content source, national broadcasting companies as aggregators, and ISPs as access providers. The need for binding agreements arose when the IOC required the broadcasting companies to ensure that (some of) the Olympic Games' content would only be delivered to receivers in the broadcasting companies' home countries (e.g., in the UK for the BBC). To accomplish this, the broadcasting companies set up binding agreements with ISPs that were known to only serve that type of users.

Besides networks, a binding agreement can for instance contain the type of transport that an aggregator has to use to deliver channels via the access provider's networks and the access provider's long-term traffic statistics (e.g., the typical traffic load at a certain time of day).

2.5 Related Work

Business networks for the distribution of multimedia channels already exist [Vernick01, TINA97, DOLMEN98], but as far as we know none of them enable users to switch between different aggregators and receive channels in different configurations. In addition, the systems that are similar to the ALIVE system (e.g., [Dutta02, Trossen03, Roy02]) are often not based on a particular business network. Even if they are, the business network does not cover the agreements between the roles in the network. We refer to Section 3.8 for a detailed discussion on the differences between our business network and the ones used by systems similar to the ALIVE system.

2.6 Summary

The ALIVE business model describes the possible relations that can exist between domains that are involved in the distribution of multimedia channels. The key characteristic of the business model is that it allows content sources to distribute the same channel via multiple aggregators. This enables mobile users to receive the same channel from multiple alternative aggregators, which increases the users' flexibility. At the same time, the use of aggregators also offers scalability and cost advantages to content sources. The costs of using (multiple) aggregators is that they increase the complexity of the Internet infrastructure.

The users in the ALIVE business model typically set up a delivery agreement (a subscription) with one aggregator, which frees them from having to establish agreements with potentially many individual sources. The distinctive agreement in the ALIVE business model is an application-level roaming agreement, which is an agreement that aggregators establish amongst each other to enable users to get access to multiple aggregators. The aggregator with which a user established a delivery agreement is that user's home aggregator, while all other aggregators are foreign aggregators.

The aggregators in the ALIVE business network offer channels in multiple configurations to serve different types of mobile hosts over different types of networks. The configurations that an aggregator supports typically provide different perceptual qualities and require different amounts of resources (e.g., network bandwidth). The configurations in the ALIVE business network typically sample the 'configuration spectrum' in a course way, thus striking a balance between a one-configuration-for-all and individual per-user configurations (e.g., fine-tuned to the user's instantaneously available bandwidth).

A delivery agreement defines in which configurations a user is allowed to receive channels from his home aggregator. We call these configurations a user's allowed configurations. Roaming agreements define in which configurations a user can receive channels from foreign aggregators, which may differ from the configurations of the user's home aggregator. As a result, a user's set of allowed configurations can differ from aggregator to aggregator.

The application and network-level parts of the ALIVE business network are largely independent of each other, except when an aggregator is linked to an access provider through a binding agreement. A mobile host that receives a channel through a local aggregator must switch to another aggregator when it leaves its current aggregator's service area.

In the rest of this thesis, we concentrate on the 'front-end' of the ALIVE business network, in particular on its application-level part (i.e., aggregators and users). We furthermore focus on the signaling interactions between

mobile hosts and aggregators and do not consider the specifics of the multimedia content itself (e.g., in terms of packet forwarding, compression, and packetization mechanisms).

Chapter 3

The ALIVE System[3]

This chapter discusses the design of the ALIVE system. The system's key feature is its ability to dynamically switch a mobile host to the aggregator that provides a particular channel in the best configuration. The ALIVE system can execute these switches automatically, thus hiding the complexity of the aggregator and network infrastructure from end-users.

The ALIVE system is designed with scalability in mind because it has to be capable of serving a large number of mobile receivers. This for instance means that the design puts a large part of the system's intelligence on mobile hosts, which is in line with current Internet design principles. The ALIVE system includes an application-level protocol, which we implemented using the Session Initiation Protocol (SIP) and the Session Description Protocol (SDP).

Before delving into the details, we first provide an overview of the ALIVE system (Section 3.1). After that, we consider the system's architecture (Section 3.2), its end-to-end interactions (Section 3.3), and internal organization (Section 3.4). Next, we take a look at the policies used by mobile hosts to take switching decisions (Section 3.5) and consider the ALIVE protocol (Section 3.6) and its implementation (Section 3.7). We conclude this chapter with a comparison of the ALIVE system with similar systems (Section 3.8).

3.1 Overview

This section provides an overview of the ALIVE system. We discuss the system's goal (Section 3.1.1) and the ways in which it should be able to move mobile hosts between aggregators (Section 3.1.2). Next, we consider the capabilities the system requires to automatically execute switches (Section 3.1.3), and the non-functional properties it needs to possess (Section 3.1.4).

---

[3] This chapter is based on [Hesselman03], [Hesselman05], and [Kamilova05].

3.1.1 Goal of the ALIVE System

The goal of the ALIVE system is to *dynamically* exploit the availability of multiple alternative content aggregators in a user-friendly manner. To accomplish this, the ALIVE system *automatically switches* mobile hosts to the aggregator that provides a channel in the best configuration (e.g., the one that provides the highest quality at a certain price) *while* they receive that channel. As a result, mobile hosts alternately receive a channel from different aggregators in different configurations (e.g., [Dutta02, Roy02, Trossen03]). In this thesis, the meaning of the 'best configuration' is defined by the end-user (e.g., [Kamilova05, Wang99]), who could for instance consider the aggregator that provides the cheapest configuration of a channel the best one.

Example

Figure 3-1 shows an example in which the ALIVE system switches Bob's mobile host to another aggregator while receiving channel CNN TV (the example is an extension of Figure 2-11).

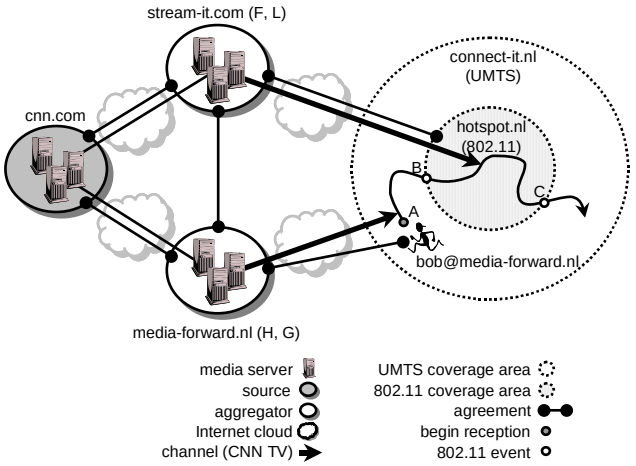


Figure 3-1. Roaming scenario.

Bob's mobile host initially receives CNN TV from media-forward.nl via the UMTS network of connect-it.nl (as of point A). However, when Bob roams into the 802.11 network of hotspot.nl, the ALIVE system discovers that stream-it.com can deliver CNN TV in a better configuration (e.g., because Bob prefers its high quality configurations over those of media-forward.nl) and therefore switches Bob's mobile host to stream-it.com. As a result, Bob receives CNN TV in one of stream-it.com's configurations from that point on. Since stream-it.com is only available in the 802.11 network, Bob's mobile host will receive CNN TV via its 802.11 interface. For the same

reason, the ALIVE system switches the mobile host back to media-forward.nl when Bob leaves the hotspot (point C).

3.1.2 Switches and Handoffs

As a result of the split-level business network of Chapter 2, the ALIVE system must be able to independently switch a mobile host to another aggregator or transfer it to another network. In this thesis, we reserve the word *switch(ing)* for a change of aggregator, while we use the term *handoff* for a change of network.

Switches

A switch from one aggregator to another requires the ALIVE system to establish a streaming session with a media server of the target aggregator and release the session with the current media server:

A switch between aggregators consists of the establishment of a streaming session with a media server of the target aggregator and the release of the streaming session with the current media server of the current aggregator.

In this definition, a session is an application-level streaming association between a mobile host and a media server in which the media server streams multimedia packets to the mobile host.

A switch between aggregators may involve the transfer of application-level context information (e.g., the state of a predictive encoder) between the aggregators' media servers [Roy02, Trossen03], but this topic is outside the scope of this thesis.

Switching Types

In general, the target aggregator of a switch can be another aggregator or the aggregator from which the mobile host is already receiving a channel. We refer to these switching types as *inter-aggregator* and *intra-aggregator*, respectively. For intra-aggregator switches, we also distinguish inter-server and intra-server switches. In the latter case, a mobile host switches to the same media server of the same aggregator to receive a channel in another configuration. This situation is comparable to standard end-to-end mobility handling (e.g., using Mobile IP [Solomon98] or SIP [Wedlund99]) because the mobile host only has to inform the media server of the mobile host's new IP address.

The ALIVE system should support all three types of switches, preferably using the same mechanisms to keep the system as simple as possible. Our examples will however focus on inter-aggregator switches.

Handoffs

The ALIVE system hands a mobile host off to another network by connecting it to one base station (e.g., an 802.11 base station) and disconnecting it from another:

A handoff between networks consists of the establishment of a connection with a target base station and the release of a connection with the current base station. A handoff includes the establishment of IP connectivity, if necessary.

A connection in this case is an IP-level association between a mobile host and the Internet.

Handoff Types

If the target base station of a handoff belongs to another network, the mobile host will need to use another IP address to communicate via that base station. A handoff to a base station of another network is usually referred to as *macro mobility*, while handoffs between base stations that belong to the same network are referred to as *micro mobility* [Campbell00]. *Figure 3-2* shows how these handoff types can be combined with the different types of switches.

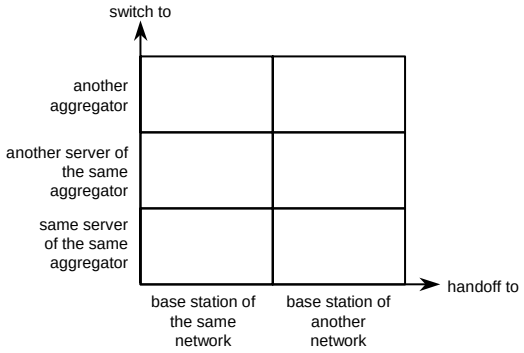


Figure 3-2. Switching and handoff types.

Handoffs can be further categorized in inter-tech handoffs and intra-tech handoffs. A handoff is called an inter-tech handoff if the target base station uses a different link-level protocol (a network technology) than the current one [Brewer98, Stemm98, Hesselman01]. For example, a handoff from a UMTS base station to an 802.11 base station is an inter-tech handoff. An intra-tech handoff, on the other hand, involves two base stations that use the same link-level protocol. Inter-tech handoffs usually require a mobile host to use another IP address on the target network (macro mobility).

In this thesis, we relax the definition of an inter-tech handoff by not requiring the ALIVE system to disconnect a mobile host from its current

base station. This enables mobile hosts to connect to multiple networks simultaneously (e.g., to the UMTS network and the 802.11 network of *Figure 3-1*), thus allowing them to exploit the availability of aggregators on multiple networks.

Finally, the target base station in a handoff can belong to the same access provider as the current one, or it can belong to another access provider. Inter-access provider handoffs typically require a change of IP address at the mobile host (macro mobility).

Macro mobility can be handled at the IP-level by protocols such as Mobile IP [Solomon98] or one its derivatives (e.g., [Tan99, Helmy00]), at the transport level [Snoeren00, Maltz98], at the 'session level' [Landfeldt99, Snoeren01], or at the application level [Wedlung99, Liao99]. Micro mobility can be handled at the level of a specific network technology (e.g., 802.11 or GPRS) or at the IP-level [Ramjee99, Campbell00]. Specific mobility handling mechanisms are however outside the scope of this thesis.

3.1.3 Automatic Switching

The ALIVE system must be able to automatically execute switches and handoffs so that users do not have to deal with the different aggregators and access providers they encounter [Kleinrock03, Latvakosi02]. The example of *Figure 3-1* illustrates that this requires the ALIVE system to be able to automatically go through five high-level steps:

1. *Detect* events (e.g., the appearance of a new 802.11 network);
2. *Decide* if a particular event might require a switch;
3. If this is the case, *discover* in which configurations a user can potentially receive a channel (e.g., CNN TV) from which aggregators;
4. *Decide* which configuration is the best one based on the user's preferences; and
5. Actually *execute* the switch.

Figure 3-3 summarizes this basic behavior as a finite state machine.

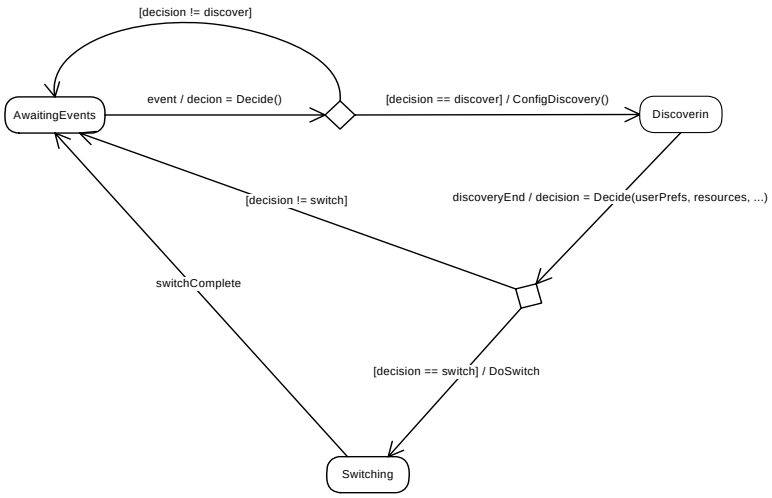


Figure 3-3. High-level behavior of the ALIVE system as a finite state machine.

Detecting Events

The ALIVE system executes switches in reaction to changes in the mobile host's environment. This means that the ALIVE system has to be able to detect events that signal such changes. Examples of events are the (dis)appearance of a network (e.g., an 802.11 network), a change in the available battery power of a mobile host, and a user changing his preferences (e.g., from 'lowest price first' to 'highest quality first').

Discovery Decisions

The decision to initiate discovery as a result of a change event generally depends on various factors. For example, if the user prefers cheap configurations, then the availability of a more expensive configuration at an aggregator will typically not make the ALIVE system decide to initiate discovery. However, if the battery power of the mobile host drops to a critical level, then the ALIVE system might decide to initiate discovery to quickly find an available configuration that requires less battery power.

Configuration Discovery

In an environment with multiple alternative aggregators, the ALIVE system has to be able to discover in which configurations a user can receive a certain channel from which aggregators. Since aggregators are access-controlled, the ALIVE system must be able to (1) authenticate users to check if they have access to a certain aggregator, and (2) authorize these users to receive a channel in certain configurations. As we have seen in Section 2.2.5, we refer to these configurations as a *user's allowed (foreign) configurations*. In general, a user's set of allowed configurations depends on a

delivery agreement and on any roaming agreements with the user's home aggregator (see Section 2.2.5). Authentication and authorization typically also involves accounting [Calhoun03], but we will not consider accounting in this thesis.

To determine which allowed configurations are actually available to a particular user, the ALIVE system also needs to check for available resources (e.g., the battery power of mobile hosts and the processing load on media servers). We call these a user's *available configurations* (for a certain aggregator and channel):

An available configuration is an allowed configuration in which a user can actually receive a particular channel from a certain aggregator.

Similar to allowed configurations, the total set of available configurations of a particular user can consist of multiple sets of available configurations of multiple aggregators (see Section 2.2.5). *Figure 3-4* illustrates this. The circles in *Figure 3-4* represent sets of configurations at the two aggregators of *Figure 3-1*.

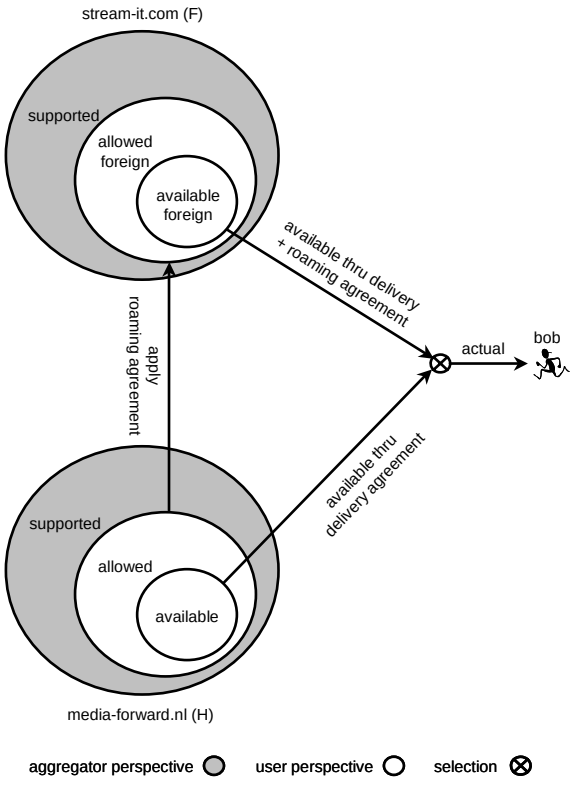


Figure 3-4. Multiple sets of available configurations at multiple aggregators.

As for available bandwidth, the ALIVE system only checks the maximum MAC level capacity (e.g., 11, 5.5, 2, or 1 Mbps for 802.11b networks) and does not measure the instantaneously available bandwidth. This limits the complexity of the system, but implies that the ALIVE system as a whole provides a *best-effort* service.

Observe that the ALIVE system may need to execute a handoff to discover available configurations on a network that the mobile host currently does not connect to. For example, if Bob were to roam from the 802.11 network of hotspot.nl into the 802.11 network of another provider, the ALIVE system would first need to execute a handoff on the 802.11 interface of Bob's mobile host before it can discover available configurations on the target network. In a situation like this, the ALIVE system also needs to discover the available local aggregators on the target network and perhaps discover their capabilities (e.g., if they automatically report the availability of new configurations).

In the future, software radios [Moessner02] might make handoffs during configuration discovery unnecessary. The ALIVE system could for instance use one of a mobile host's physical interfaces to continuously discover available configurations on different types of networks, while at the same time using another physical interface to actually receive a channel. This would enable the ALIVE system to discover configurations 'in the background' and only execute handoffs as a result of an actual switch. The use of software radios is however outside the scope of this thesis.

Switching Decisions

The ALIVE system has to decide (1) which aggregator can deliver a channel in the best available configuration, (2) which of that aggregator's media servers will deliver the channel in that configuration, and (3) how the switch needs to be executed (e.g., in a make-before-break manner by first establishing a streaming session with one of the target aggregator's media servers and then tearing down the session with the media server of the current aggregator).

Another requirement is that users and other stakeholders (e.g., the owner of the mobile host or its manufacturer) should be able to flexibly change the rules based on which the ALIVE system makes discovery and switching decisions. It should be possible to change these rules while the system is in operation, so that the entire system can remain 'always on'. For example, being able to change the ALIVE system's switching rules enables mobile users to change their preferences. It also allows the owner of a pool of mobile hosts to control the resources these hosts consume for aggregator switching (e.g., by not allowing the ALIVE system to execute switches in a make-before-break manner, which is usually more expensive because it uses more resources).

In this thesis, we concentrate on the discovery and switching rules that define the behavior of mobile hosts and aggregators. We do not consider the rules that define the operation of access providers (e.g., when they admit users to their networks).

Executing a Switch

Switches in the ALIVE system can take place in two ways: in a make-before-break manner or in a break-before-maker manner. In a make-before-break switch, the ALIVE system first establishes a multimedia streaming session between the mobile host and the target media server of the target aggregator, and then releases the session between the mobile host and the current media server. This is similar to make-before-break handoffs in mobile networks, which for instance occur in overlay situations [Stemm98, Brewer98, Hesselman01]. In a break-before-make switch, the ALIVE system first releases the session with the current media server and then establishes the session with the target media server. Break-before-make handoffs for instance occur in 802.11 networks [DeCleyn04].

The result of a switch is that a mobile host receives a channel in the best available configuration, possibly via another network than before the switch (cf. the example of *Figure 3-1*). We call this available configuration the actual configuration:

An actual configuration is the best available configuration and is the configuration in which a mobile host actually receives a particular channel.

Initial and Final Switches

To begin the playout of a particular channel at a mobile host, the ALIVE system has to perform an initial switch, which is a switch from a 'null' aggregator to an initial target aggregator. Similarly, to terminate a channel the ALIVE system needs to switch a host from its current aggregator to a 'null' target aggregator. In this thesis, we will however not consider these switches. Instead, we will assume that a mobile host is already receiving a channel before a switch and will continue to do so after the switch. Applications like Electronic Program Guides (cf. the Mbone tool sdr [SDR]) that enable a user to select a channel are therefore outside the scope of this thesis as well.

3.1.4 Non-Functional Requirements

The design of the ALIVE system is based on three non-functional requirements:

- The system should be able to switch a mobile host to another aggregator before the mobile host runs out of multimedia packets to render. As a result, switches will take place in a smooth manner (i.e., without glitches), thus hiding them from the user. To achieve this goal, the ALIVE should for instance perform certain tasks in parallel, for example the discovery of available configurations on different networks and the discovery of new aggregators on these networks;
- The system should minimize the amount of control information transferred to and from mobile host to save bandwidth, which is generally scarce in a wireless environment; and
- The system should be scalable. In our work, scalable means that the amount of inter-aggregator traffic and the authentication load on home aggregators should be minimized (cf. [Rosenberg98]). Following Internet design principles [Saltzer84, Clark88], it also means that aggregators should maintain the minimum possible amount of state. As a result, a large part of the system's intelligence will reside on mobile hosts.

3.2 ALIVE Architecture

Figure 3-5 shows the high-level architecture of the ALIVE system superimposed on the example of *Figure 3-1*. The architecture's main functional components are *switching controllers* on mobile hosts, signaling *front-ends* at aggregators, and *aggregator directories* at access providers. Note that *Figure 3-5* represents access providers as IP clouds rather than radio coverage areas (cf. *Figure 3-1*).

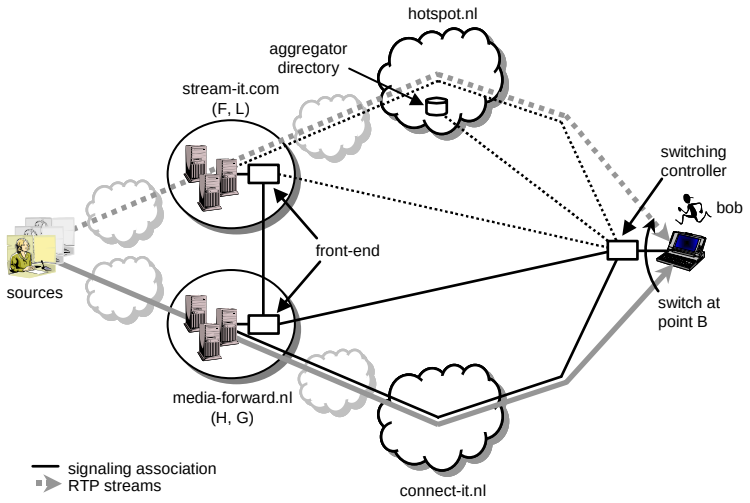


Figure 3-5. High-level system architecture superimposed on the example of *Figure 3-1*.

The rest of this section provides an overview of the ALIVE system architecture. We first consider the architectures of mobile hosts (Section 3.2.1), aggregators (Section 3.2.2), and access providers (Section 3.2.3). After that, we take a look at the signaling associations between the switching controllers of mobile hosts, the front-ends of aggregators, and the aggregator directories of access providers (Section 3.2.4). The interactions that take place on these signaling associations are the subject of Section 3.3.

3.2.1 Host Architecture

Figure 3-6 shows the architecture of the mobile hosts in the ALIVE system. The host's switching controller is the center of the architecture. Switching controllers typically interact with multiple front-ends (Section 3.2.2) and multiple aggregator directories (Section 3.2.3).

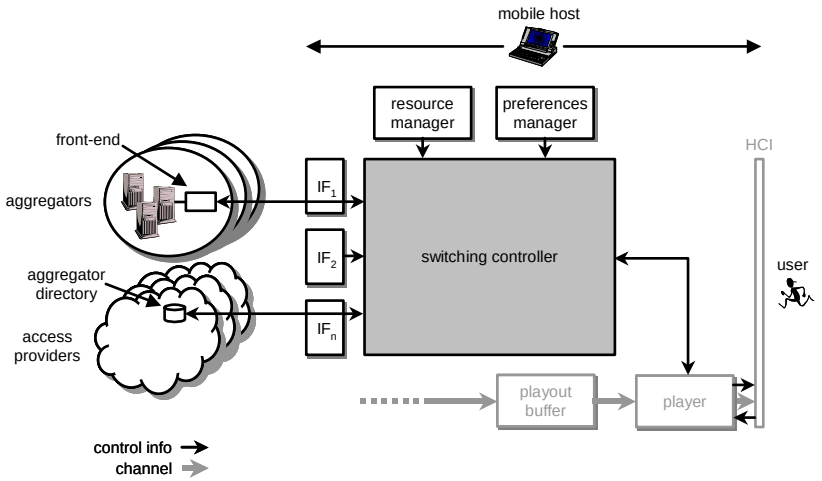


Figure 3-6. Architecture of mobile hosts in the ALIVE system.

Switching Controller

In the ALIVE system, most of the responsibilities for automatic switching lie with the switching controllers, which means that they go through the five high-level steps of Section 3.1.3. As a result, the switches in the ALIVE system are *mobile-controlled* (cf. mobile-controlled handoffs between networks [Tripathi98]). The advantage of mobile-controlled switching is that it moves most of the system logic to the mobile host, which is in line with current Internet design principles [Clark88, Saltzer84]. The mobile host is furthermore a natural place to control switches because it is typically aware of the aggregators it can reach from a particular location. An alternative approach is to off-load parts of the switching controller's functions to aggregators. For example, the part of the switching controller

responsible for making switching decisions (e.g., when to discovery available configurations and which aggregator will provide the actual configuration) could be located at an aggregator, while the rest of an switching controller's functions could remain at the mobile host (cf. network-assisted handoffs [Tripathi98]). This approach requires the switching controller to upload certain information that allows the aggregator to make these decisions, such as the available processing power on the mobile host, and available codecs and session control protocols [Xu00]. Such an approach may be advantageous for resource constrained mobile hosts, but it requires aggregators to deal with switches for a potentially large number of mobile hosts. In addition, it may be difficult to have a single front-end decide on an actual configuration in an environment with multiple aggregators.

Observe that a switching controller is a *cross-layer* component because it is responsible for executing switches (at the application-level) and for executing handoffs (at the network-level).

Information Sources

A switching controller uses several information sources to detect changes in the mobile host's environment (e.g., the availability of a new network on a certain interface) and to make decisions (e.g., to decide which available configuration a user prefers). In this thesis, we distinguish five information sources:

- A *local resource manager*, which keeps track of the available local resources on the mobile host (e.g., battery power and available codecs);
- A *preferences manager*, which manages the user's preferences (e.g., regarding quality and costs);
- *Network interfaces* (e.g., 802.11 and UMTS interfaces) through which the switching controller interacts with front-ends and through which the mobile host actually receives a channel. A network interface also keeps track of information such as the networks currently available on an interface, their signal strengths, and so on;
- *Front-ends* (remote sources), which enable the switching controller to discover the available configurations in which a user can receive a particular channel (see Section 3.2.2); and
- *Aggregator directories* (remote sources), which enable switching controllers to discover which local aggregators are available through a certain network (see 3.2.3).

In general, the switching controller can regularly poll these components for changes or it can wait for them to push such changes to it. For example, a switching controller can actively poll a front-end for it available configurations, or it can simply wait for the front-end to generate a notification that signals a change (through a configuration notification

message). Of course, the latter requires front-ends to have the capability to generate configuration notifications.

In this thesis, we only *use* the information provided by the local resource manager, the preferences manager, and the network interfaces. We do not consider the internals of these components.

Multimedia Components

The other components on a mobile host deal with multimedia streams:

- A multimedia *player* that depacketizes, decompresses, and renders the multimedia information in a multimedia stream. The player also provides an interface to the user than enables him to control the playout of a channel (e.g., pause, stop, change quality); and
- A *playout buffer.* The playout buffer receives multimedia packets from the network and temporarily stores them to ensure that it can continue to feed information to the player when the packets of a multimedia stream are lost [Karrer01] or delayed [Li02]. Since the ALIVE system is about one-way streaming, the delay buffer can be quite deep (e.g., approximately 45 seconds for RealPlayer [Li02]). The buffer typically stores RTP packets [Schulzrinne96a].

The details of the player and the playout buffer are outside the scope of this thesis.

3.2.2 Aggregator Architecture

Figure 3-7 shows the architecture of an aggregator. The main component is a front-end. A front-end typically interacts with multiple switching controllers.

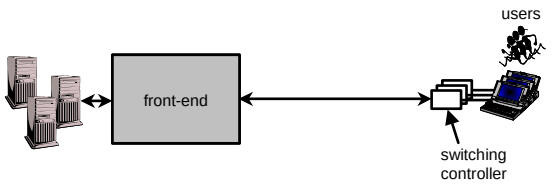


Figure 3-7. Architecture of aggregators in the ALIVE system.

Front-end

The main task of a front-end is to enable switching controllers to discover a user's available configurations for a particular channel. To accomplish this, a front-end's task in the ALIVE system is to:

- *Authenticate* mobile users;
- *Authorize* a user to receive a channel in a certain set of allowed configurations (see Section 2.2.5). For foreign users, this is the set of allowed foreign configurations, which is based on the user's delivery

agreement with his home aggregator and the roaming agreement between the home and the foreign aggregator;
- *Decide* which of a user's allowed configurations are actually available. This for instance depends on the available resources on the aggregator's media servers and on the policies of the aggregator (e.g., e.g., a policy that limits the availability of high-end configurations to off-rush hour periods); and
- *Decide* on a set of media servers from which the aggregator can deliver a channel in an available configuration.

Centralized versus Distributed Front-ends

The ALIVE system uses centralized front-ends. The advantage of this approach is that mobile hosts can get the information they need from an aggregator at a single point, which reduces the number of interactions between mobile hosts and aggregators. The disadvantage is that the front-end might form a single point of failure and that it can potentially become an aggregator's bottleneck if it is not dimensioned well (e.g., if it is not replicated across multiple machines).

An alternative approach is to distribute a front-end across multiple machines. In the most extreme case, a front-end could be fully distributed across an aggregator's media servers without any inter-media server synchronization (cf. [Amir98]). In a unicast environment, this would mean that a mobile hosts might need to communicate with many front-ends, especially when the host can reach multiple aggregators from its current location. This results in increased bandwidth consumption because mobile hosts need to send out more requests and because different front-ends of the same aggregator may report the availability of the same configuration. Another problem with this approach is that different front-ends might attempt to authenticate the same foreign user, which would increase the load on home aggregators and would decrease scalability.

A fully distributed approach might be more viable in an IP multicast environment. In that case, a mobile host could for instance use an aggregator-specific multicast group to simultaneously interact with multiple front-ends of the same aggregator. A technique like multicast damping [Amir98] could then suppress duplicate responses from front-ends that include the same available configuration. IP multicast does however not match the user-specific nature of a user's set of available configurations and is not very widespread at this point [Chennikara02].

3.2.3 Access Provider Architecture

The most relevant component of an access provider for the ALIVE system is the aggregator directory. An aggregator directory enables switching

controllers to discover which local aggregators they can reach through a certain access provider. All access providers that have a binding agreement with one or more aggregators (see Section 2.4.2) operate an aggregator directory.

3.2.4 Signaling Associations

The ALIVE system involves several signaling associations (see *Figure 3-5*). A switching controller has signaling associations with front-ends, media servers, and aggregator directories, while front-ends are also involved in signaling associations with other front-ends.

Switching Controllers – Front-ends

The purpose of the signaling association between a switching controller and a front-end is to enable the switching controller to retrieve a description of the available configurations in which a user can receive a channel. The signaling association also allows the switching controller to authenticate a user with the front-end and to retrieve a description of the front-end's capabilities. We refer to Section 3.3 for a discussion on the interactions that take place on this signaling association.

Switching Controllers – Media Servers

The purpose of the signaling associations between switching controllers and media servers is to execute a switch. The signaling association enables switching controllers to establish a streaming session with a media server of the target aggregator and to release the streaming session with the current media server (see Section 3.1.2).

The advantage of direct signaling paths between switching controllers and media servers is that it simplifies front-ends. For example, front-ends do not need to act as proxies that establish or release a multimedia session between a media server and a mobile host on behalf of the switching controller. This would require front-ends to be more intelligent, which goes against the design goals of Section 3.1.4.

However, turning front-ends into signaling proxies enables a front-end to act as a signaling gateway, for instance by translating SIP and WindowsMedia messages from mobile hosts into RTSP messages that the media servers require. This will enable aggregators to serve a heterogeneous population of mobile hosts with a homogeneous pool of servers. This is also the disadvantage of the direct signaling paths between switching controllers and media servers in the ALIVE system: it requires aggregators to operate a heterogeneous pool of media servers (e.g., with RTSP, SIP, Real, and WindowsMedia servers) to be able to serve different types of mobile hosts, some of which might only be able to deal with one type of media server.

We refer to Section 3.3 for a discussion on the interactions that take place on the signaling associations between switching controllers and media servers.

Switching Controllers – Aggregator Directories

The goal of the signaling association between a switching controller and an aggregator directory is to enable the switching controller to discover the local aggregators that are available through a certain network. The result typically consists of a set of URIs that point to the front-ends of the local aggregators.

In this thesis, we assume that the interactions on this signaling association take the form of DHCP messages [Droms99, Vatn98] and that aggregator directories use a DHCP option to convey the URIs of local aggregators to the switching controller (e.g., using the DHCP option for SIP URIs [Schulzrinne02]). Alternative protocols that can be used are the Service Location Protocol (SLP) [Guttman99] or the telephony gateway location protocol discussed in [Rosenberg98].

Front-ends – Front-ends

The purpose of the associations between front-ends is to enable aggregators to authenticate foreign users with their home aggregators. Two front-ends establish a signaling association when there exists a roaming agreement between the two aggregators (cf. the signaling links between federated UMTS domains [3GPP99]). The front-end of the foreign aggregator uses the association to authenticate foreign users at their home aggregator and to retrieve a description of their set of *allowed* configurations. An alternative approach is to retrieve so-called authentication vectors from the home aggregator and authenticate the user at the foreign aggregator, as is the case in UMTS [Køien03].

In this thesis, we assume that the interactions on the signaling association between front-ends are based on a AAA protocol like Diameter [Calhoun03]. The specifics of these interactions are however outside the scope of this thesis.

Signaling Associations and Network Interfaces

In the ALIVE system, a signaling association with a local aggregator must be established via a network to which the local aggregator is bound (through a binding agreements, see Section 2.4.2). This is necessary because the services that a local aggregator offers are unavailable via networks that are not part of its service area. For example, stream-it.com (*Figure 3-1*) only accepts packets sent through hotspot.nl's 802.11 network and blocks packets that originate from other networks (a similar firewalled setting is discussed in [Hsieh03]).

In general, switching controllers may be able to establish multiple concurrent signaling associations with the same front-end, typically via multiple networks. For example, inside the hotspot of *Figure 3-1*, the switching controller on Bob's mobile host could set up a signaling association with multimedia-forward.nl via the host's UMTS interface as well as via its 802.11 interface (multimedia-forward.nl is a global aggregator).

To simplify the ALIVE system, we assume that a switching controller establishes at most one signaling association at a time with a certain front-end. As a result, each signaling association (with front-ends of local aggregators or with front-ends of global aggregators) is bound to one network interface. All messages that the signaling association carries enter/leave the mobile host via that interface. For example, the signaling association between the switching controller on Bob's mobile host and the front-end of media-forward.nl is either bound to the UMTS network or to the 802.11 network, but not to both.

3.3 End-to-end Interactions

This section discusses the high-level end-to-end interactions that take place on the signaling associations between switching controllers and front-ends as well as between switching controllers and media servers (see Section 3.2.4). These interactions are: authentication interactions (Section 3.3.1), configuration discovery interactions (Section 3.3.2), capability discovery interactions (Section 3.3.3) and switching interactions (Section 3.3.4). The latter take place on the signaling association between switching controllers and media servers.

3.3.1 Authentication

An authentication interaction enables a switching controller to authenticate a user with a front-end. In the ALIVE system, a front-end must have been able to authenticate a user before it allows a switching controller to make use of its services.

Caching Authentication State

In the ALIVE system, successful authentication results in the switching controller receiving a cryptographic *token* that it has to use in further communications with the front-end (e.g., to discover an aggregator's available configurations). A token indicates that the front-end has *cached* the user's authentication state, which enables the front-end to reauthenticate the user from its cache rather than at the user's home aggregator. This

reduces the amount of traffic in the Internet as well as the load on home aggregators, which aids the scalability of the system [Rosenberg98]. It also reduces the per-interaction delay between the switching controller and a front-end, which enables mobile hosts to switch between aggregators more quickly. Similar authentication caches are used in 802.11 networks to pre-authenticate a user with a set of target access points to which the user's mobile host can potentially handoff [Mishra04, Pack02].

The token may need to be protected from eaves dropping, for instance through a mechanism that uses a pre-defined key to automatically change the token at both ends on an association after each interaction [Mishra04] or through an encryption technique such as IPsec. Such security mechanisms are however outside the scope of this thesis.

Refreshing Authentication Softstate

In the ALIVE system, front-ends cache a user's authentication state as softstate, which means that switching controllers need to regularly refresh this state. A switching controller and a front-end negotiate a suitable refresh interval during authentication.

A switching controller can use its own refresh interval for each individual aggregator. Each of these intervals is however constrained by the range of refresh intervals acceptable to the respective front-ends. The rationale behind this approach is that switching controllers will typically strive for a long refresh interval because it reduces their bandwidth consumption and costs, and because it saves battery power. Front-ends, on the other hand, will typically use the length of the refresh interval to trade off memory usage (a longer refresh interval will require front-ends to maintain more authentication state) and the bandwidth required to handle refresh requests (a shorter refresh interval will result in the arrival of more refresh request messages).

3.3.2 Configuration Discovery

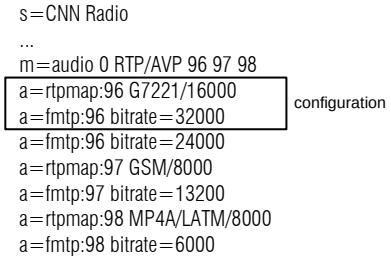
Configuration discovery interactions enable switching controllers to discover in which available configurations a user can receive a channel from an aggregator. To accomplish this, front-ends must describe their available configurations. Configuration discovery interactions have to carry the authentication token of Section 3.3.1.

Configuration Descriptions

Figure 3-8 shows what the description of set of potential configurations could look like in the language of the Session Description Protocol (SDP) [Handley98]. We will use SDP in this thesis because it is an IETF standard.

An alternative description language is that of SDP Next Generation (SDPng) [Kutscher03], but this is not a standard yet.

Figure 3-8. SDP description of available configurations.



The first line in *Figure 3-8* (s=) contains the name of the channel, which is CNN Radio in this example. The next line (m=) describes the media type (audio), followed by an indication that the description is based on the RTP Audio-Video Profile (AVP) [Schulzrinne96b]. An RTP profile defines how compressed data streams must be broken up into packets for transmission over the Internet. An RTP profile is codec-specific. The three numbers behind the RTP/AVP keyword are profile identifiers.

The attribute lines (a=) describe the actual configurations. The description of a single configuration consists of an rtpmap line and one of the fmtp lines that follow the rtpmap line. *Figure 3-8* thus describes four configurations, two G.7221 configurations, one GSM configuration, and one MP4A configuration. An rtpmap line describes the configuration's codec type (e.g., a G.7221 codec), while the fmtp lines describe codec specific parameters (e.g., a bit rate of 13.2 kbps for the GSM configuration).

We note that the bitrate parameters in the fmtp lines in the above example are for illustrative purposes only. In reality, these parameters are codec-specific. Also note that the payload types in the media line (m=) merely form alternatives and do not express an ordering, as is normally the case in SDP. For simplicity, we omitted all other SDP lines other than s= and m=.

Media Server URIs

A front-end includes a number of media server URIs in each description of an available configuration. The URIs point to the aggregator's media servers that can currently deliver the channel in that available configuration. The set of URIs of a single configuration description may consist of different types of URIs (e.g., a SIP and an RTSP URI). *Figure 3-9* shows an example (an extension of *Figure 3-8*) in which the URIs appear in a= lines immediately after a configuration's rtpmap and fmtp lines.

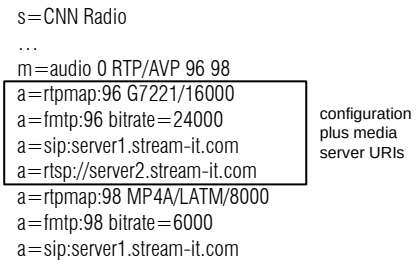


Figure 3-9. SDP description of available configurations, including media server URIs.

The advantage of including URIs a configuration description is that switching controllers can directly establish a streaming session with one of the aggregator's servers (e.g., using a protocol like SIP [Rosenberg02a] or RTSP [Schulzrinne98]). This minimizes the number of interactions between the switching controller and a front-end, which benefits the speed at which the ALIVE system can switch mobile hosts to another aggregator. It also enables mobile hosts to immediately start an appropriate media player based on the type of the URI (e.g., a media player that supports SIP for a SIP URI). This is important when a mobile host switches between different types of media servers, for instance from an RTSP server to a SIP server. The downside is that it requires the media servers of an aggregator to interact with the aggregator's front-end, for instance to signal that a user has begun to receive a channel.

Observe that a switching controller will be involved in selecting a media server if a configuration description contains multiple URIs. In this case, sever selection is a function that is distributed across front-ends and switching controllers.

Virtual Home Environment

The aggregators in the ALIVE system provide a *virtual environment of home configurations*. This means that front-ends use the configurations in a user's delivery agreement (i.e., his allowed configurations) to describe a user's available configurations, irrespective of whether the user is a foreign user or a user that has a delivery agreement with the aggregator. At the same time, aggregators actually deliver channels in their own configurations. For example, the front-end of stream-it.com (*Figure 3-5*) describes the CNN TV configurations available to Bob in terms of the configurations that appear in Bob's delivery agreement with media-forward.nl, but stream-it.com's media servers deliver CNN TV in one of its own equivalent configurations. *Figure 3-10* illustrates this.

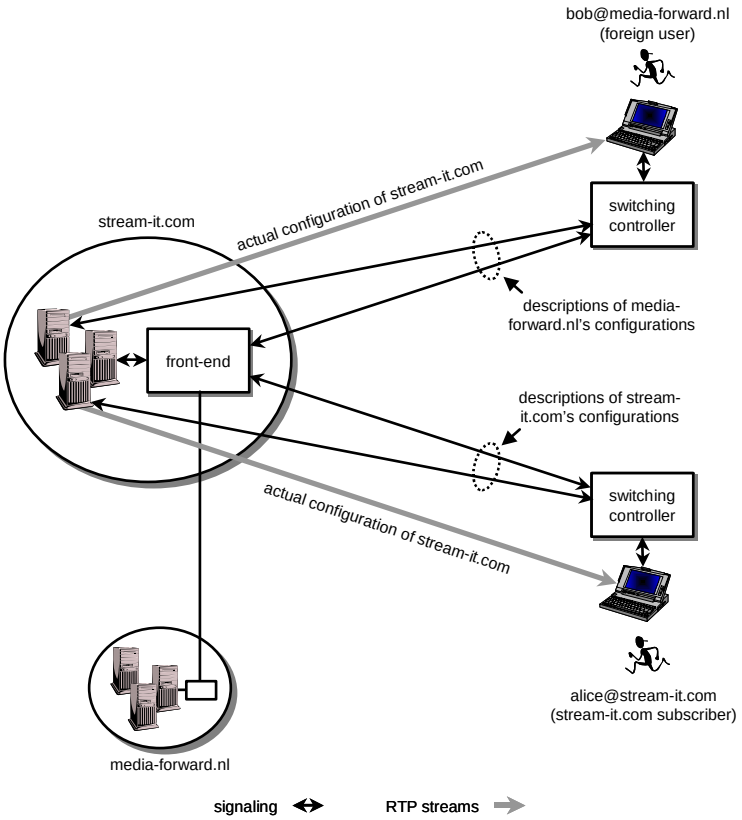


Figure 3-10. Virtual environment of home configurations.

The main advantage of a virtual home environment is that foreign aggregators do not have to expose descriptions of the configurations they support to foreign users, which is something that they may consider undesirable for competitive reasons. Another advantage is that it simplifies switching controllers because they only have to deal with the configuration descriptions of a user's home aggregator. This for instance simplifies the mapping between configuration descriptions and the quality label associated with the configuration (see Section 2.2.4). A disadvantage of the approach is that the perceptual quality of the foreign configurations may slightly differ from those of the home configurations, which may be noticeable to the user (who expects a quality levels of the configurations of his home aggregator). In addition, aggregators need to posses more intelligence because front-ends need to be able to map foreign configurations to the configurations of the user's home aggregator and media servers need to do the opposite (see Section 3.3.4). The conversion will however be computational simple. Roaming agreements furthermore typically change infrequently, which for

instance enables media servers to store roaming agreements locally and do the conversion themselves.

Observe that the virtual home environment of configurations is similar to the ubiquitous availability of certain service numbers in contemporary cellular networks (e.g., 333 for voice mail access).

Configuration Changes

Changes in a user's set of available configurations may make a switching controller decide to switch to another aggregator. One way to detect such changes is to have switching controllers poll a front-end for a description of its currently available configurations. However, a more efficient approach is to make use of an eventing mechanism in which a front-end pushes configuration change events to switching controllers (cf. the announcement protocols of SIP [Roach02], SLP [Kempf00], and UPnP [Microsoft00]). Switching controllers receive these events after they have subscribed to the notification service.

The change notifications that a front-end issues contain the name of a channel and a description of available configurations in which a user can currently receive that channel. An alternative approach is to transmit the delta with the previous notification message, which means that a notification only describes those configurations that have become (un)available since the transmission of the previous configuration notification. The downside is that this requires the switching controller to receive all the notifications that a front-end sent to maintain a consistent view of a user's available configurations, which may be difficult in a wireless environment. An advantage of only transmitting deltas is that it saves network bandwidth as the size of the messages will typically be smaller.

Preferred Configurations

As part of configuration discovery, switching controllers can express their interest in a subset of a user's allowed configurations. A switching controller could for instance prefer configurations that provide mono audio because the mobile host does not have the capabilities to deal with configurations that provide stereo audio.

3.3.3 Capability Discovery

To decide if it wants to use a particular aggregator, a switching controller may first want to discover the capabilities of its front-end. In this thesis, we only consider one capability, which is the event reporting capability. If a front-end has this capability, it can report events that signal changes in a user's set of available configurations. As a result, switching controllers do

not have to regularly send configuration requests to such front-ends (i.e., poll it).

We assume that the capabilities of front-ends change seldomly, which means that there is no need for capability notifications that signal changes in a particular front-end's capabilities.

In general, it is also possible that the different media servers that appear in a configuration description (see Section 3.3.2) have different capabilities. For example, some servers may allow switching controllers to begin the reception of a channel at a specific point (e.g., 5 minutes and 30 seconds from the beginning). Switching controllers might prefer such servers because it will enable them continue to receive a channel at exactly the same point where it left off at the old server. We will however not consider the topic of media server capabilities any further in this thesis.

3.3.4 Switching

Switching interactions involve the establishment of a multimedia session with a target media server and the release of the multimedia session with the current media server. A switching controller establishes and releases multimedia sessions through direct signaling associations with media servers (see Section 3.2.4).

To establish a multimedia session, the switching controller must inform the target media server of the user's authentication token and a description of the actual configuration in which the switching controller wants to receive a channel from the media server. As a result of the virtual environment of home configurations, the actual configuration is an allowed configuration of the user's home aggregator (see Section 3.3.2), which means that the media sever of a foreign aggregator needs to map it to one of its own configurations.

To release a session, the switching controller has to inform the media server of the user's authentication token and a description of an actual configuration. Strictly speaking, the configuration description is only required for intra-server switches (see Section 3.1.2). For all other types of switches, media servers need to be able to detect switching controllers that do not release a multimedia session in an orderly manner (e.g., using RTCP Receiver Reports [Schulzrinne96a]), for example because their network connection went down suddenly. The media servers can then clean up the resources the mobile host was using (e.g., transcoders, if any) and inform the front-end that the user is no longer receiving the channel.

3.4 ALIVE Control Points and Services

Figure 3-11 shows the internal organization of switching controllers and front-ends. Both components consist of a *control point* and an *ALIVE protocol entity* (PE in *Figure 3-11*). The control points are primarily responsible for local processing (e.g., deciding if the switching controller should initiate configuration discovery), while the ALIVE protocol entities focus on realizing the end-to-end interactions of Section 3.3 (e.g., refreshing a user's authentication state). For ease of writing, we usually refer to the control point of a switching controller as the *client control point*.

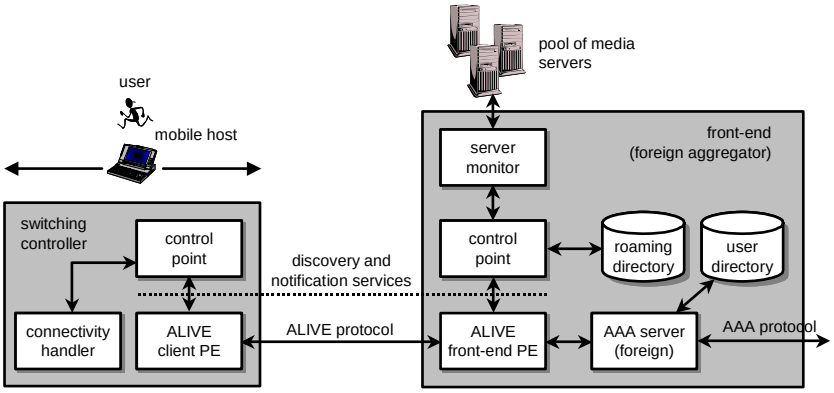


Figure 3-11. Internal organization of switching controllers and front-ends.

The ALIVE protocol entities provide four services to the control points:

- A *configuration discovery* service, which enables client control points to discover in which configurations it can receive a particular channel;
- A *capability discovery* service with which a client control point can discover the capabilities of front-ends; and
- A *configuration notification* service, which informs client control points of changes in the available configurations of a channel.

The ALIVE protocol entities do not provide a capability notification service because we have assumed that a front-end's capabilities change very infrequently (see Section 3.3.3).

A switching controller also contains a *connectivity handler*, which is a component that takes care of the network-level interactions, for instance with aggregator directories and access points (e.g., to execute a handoff). A connectivity handler offers three services:

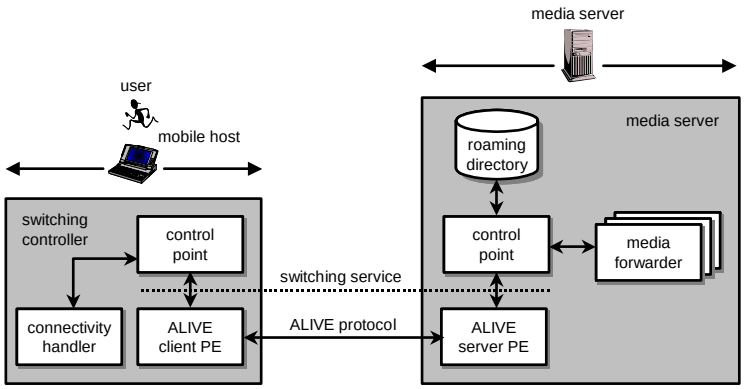
- An *aggregator discovery* service, which enables a client control point to discover aggregators;
- An *aggregator notification* service, which informs client control points of changes in the availability of aggregators on a network; and

– A *handoff* service, which enables client control points to execute a handoff.

The other components in *Figure 3-11* are a server monitor, a roaming directory, a user directory, and a AAA server. The *server monitor* is a local service that keeps track of the available resources on the aggregator's media servers. The front-end control point uses this service to determine in which configurations the aggregator's media servers can currently deliver a particular channel. The *roaming directory* stores the roaming agreements that the aggregator has established with other aggregators. The front-end control point uses the roaming directory to map descriptions of its own configurations to foreign configurations, and vice versa (see the virtual home environment of Section 3.3.2). The AAA server and the user directory are used by the ALIVE protocol entity at the front-end and will be discussed Section 3.6.3.

Figure 3-12 shows the internal organization of a media server, which also contains a control point and an ALIVE protocol entity. The ALIVE protocol entity on the mobile host and the ALIVE protocol entity on the media server provide a *switching* service, which enables a client control point to execute a switch.

Figure 3-12. Internal organization of media servers.



The other components on a media server are a roaming directory and a set of media forwarders. The roaming directory contains the same information as the roaming directory of the front-end. The control point of the media server uses the roaming directory to map descriptions of the aggregator's own configurations to foreign configurations, and vice versa (virtual home environment). The *media forwarders* receive the actual multimedia streams from a source, optionally manipulate them, and forward them to a mobile host.

In this section, we focus on the behavior of the control points and on the services provided by the ALIVE protocol entities and the connectivity

handler. We assume that these services offer some form of transaction management (e.g., a transaction ID in the services' primitives) so that the control points can match requests with responses.

We first consider the control points of switching controllers (Section 3.4.1), front-ends (Section 3.4.2), and media servers (Section 3.4.3). After that, we consider the configuration discovery service (Section3.4.4), the configuration notification service (Section 3.4.5), the capability discovery service (Section 3.4.5), and the switching service (Section 3.4.7). At the end of this section, we also consider the services of the connectivity handler (Section 3.4.8) and the server monitor (Section 3.4.9). The details of these two components are however outside the scope of this thesis, which is why we only consider the local services that they provide.

We will consider the ALIVE protocol and the internal behavior of the ALIVE protocol entities in Section 3.5.

3.4.1 Client Control Point

The control point on the mobile host goes through the five steps of Section 3.1.3. It uses the services provided by ALIVE protocol entities as follows:

1. The control point detects remote environment changes through the configuration notification service and the aggregator notification service. Local components such as the resource manager (see Section 3.2.1) also inform the control point of environment changes on the mobile host (e.g., a change in available battery power);
2. If configuration discovery is necessary, the control point invokes the configuration discovery service. During configuration discovery, the control point might use the handoff service to execute a handoff on one of the host's interfaces. After such a handoff, the control point invokes the aggregator discovery service to discover new local aggregators on the target network and the capability discovery service to discover the capabilities of these front-ends (optional). If the control point decides to use any of the new aggregators, it reinvokes the configuration discovery service for the newly discovered aggregators. The latter implies that the configuration discovery service should be able to add new aggregators to an ongoing discovery procedure; and
3. If a switch is required, the control point invokes the switching service.

Hysteresis

The client control point should apply some sort of hysteresis to the environment changes. An 802.11 network interface could for instance alternately report that the 802.11 signal strength is 'low' or 'moderate', which may have the effect that the control point ping-pongs between

aggregators [Hesselman01]. A similar problem exists at the network-level where mobile hosts handoff between networks [Pollini96].

Parallel Tasks

The services provided by the ALIVE protocol entities and the connectivity handler should enable the client control point to perform certain tasks in parallel to minimize the configuration discovery delay. Specifically, they should enable the client control point to

- Simultaneously discover available configurations on different interfaces;
- Discover new aggregators on a new network (i.e., after a handoff) and concurrently discover the capabilities of their front-ends; and
- Decide which available configuration is the best one during discovery.

3.4.2 Aggregator Control Point

The control point of a front-end is the peer of a client control point. A front-end's control point governs the availability of an aggregator's configurations. It has two responsibilities:

- Make availability decisions, which means that the control point decides which of the aggregator's supported configurations are available configurations (e.g., using the aggregator's policies); and
- Map descriptions of the aggregator's supported configurations to descriptions of foreign configurations, and vice versa.

Authentication and authorization of users is transparently handled by the underlying ALIVE protocol entity (see Section 3.6), which enables the control point to concentrate on the availability of the aggregator's configurations.

We distinguish two types of events that require an availability decision: local events (e.g., the time of day changing to rush-hour or the server monitor indicating that the available resources on the media servers have changed) and remote events, specifically requests from client control points for a description of a channel's available configurations.

Requests from Client Control Points

If the configuration discovery service indicates that a client control point solicits a description of the available configurations of a channel, the front-end control point goes through three steps:

1. Decide in which configurations the aggregator can currently deliver the channel to the user. This requires the control point to get a description of the user's allowed configurations and consult the server monitor to check in which configurations the aggregator's media servers can

currently deliver the channel. The control point also needs to make a decision on the media servers that it will present to the client control point for each available configuration;
2. Use the ID of the user's home aggregator and the roaming agreements in the roaming directory to map the descriptions of the available configurations to descriptions of equivalent foreign configurations (this step is null if the user is not a foreign user); and
3. Use the configuration discovery service to return the result to the client control point.

Local Events

For local event, the control point also goes through three steps:
1. Decide which channels are available in which configurations and check if this was different before the event;
2. If the available configurations of a particular channel have changed, use the roaming agreements to map the new set of available configurations to foreign configurations. Do this for every foreign aggregator that appears in the roaming directory; and
3. Use the configuration notification service to convey the descriptions of the foreign configurations to client control points.

3.4.3 Media Server Control Point

The control point of a media server is a peer of the client control point on a mobile host. A media server control point establishes and releases multimedia sessions with a mobile host. It has two responsibilities:

- Start and stop media forwarders; and
- Map descriptions of the aggregator's supported configurations to descriptions of foreign configurations, and vice versa (also see front-end control point, Section 3.4.2)

The control point of a media server receives requests for the establishment or release of a session from the switching service. The requests contain the name of a channel and a description of a configuration (see Section 3.4.7).

The behavior of a media server's control point to establish a multimedia session consists of three steps:
1. If the switching service indicates that a switching controller is soliciting the establishment of a session, use the agreements in the roaming directory to translate the foreign configuration in the solicitation to one of the aggregator's own supported configurations (this step is null for users that are not foreign users);
2. Find a media forwarder that can deliver the channel in that configuration and check if there are sufficient resources to run it; and

3. If this is the case, start the media server, and inform the remote control point that the session has been established.

When the switching service indicates that a switching controller is soliciting the release of a session, the control point simply stops the corresponding media forwarder.

For each of its active media forwarders, the media server control point also checks if the forwarder's streams are still being received. If this is no longer the case (e.g., because the mobile host moved to another network without having been able to release the session), then the control point stops the corresponding media forwarder. A control point can for instance detect the disappearance of a receiver through the absence of RTCP Receiver Reports [Schulzrinne96a].

3.4.4 Configuration Discovery Service

The configuration discovery service enables the control point on the mobile host to discover the available configurations in which a user can receive a particular channel from a set of front-ends.

Figure 3-13 shows the basic behavior of the configuration discovery service. It involves four types of primitives: requests, indications, responses, and confirmations. The vertical lines in *Figure 3-13* correspond to the service boundary between a control point and an ALIVE protocol entity. Time progresses from top to bottom.

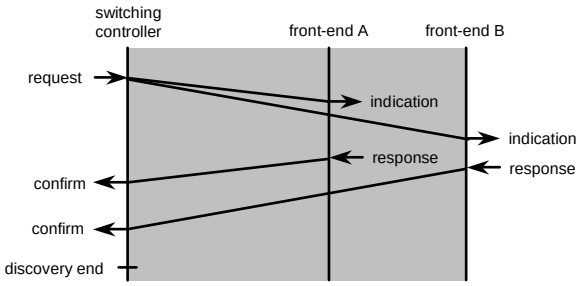


Figure 3-13. Behavior of the configuration discovery service.

The configuration discovery service is a multiparty service, which means that a single request can result in indications at multiple front-ends. This enables the control point on the mobile host to discover configurations at multiple front-ends simultaneously, which reduces the switching delay. However, the configuration discovery service reports one confirmation per front-end (i.e., per response primitive) and issues them *during* discovery (i.e., instead of issuing a batch confirmation at the end of the discovery period). This enables the control point to make switching decisions as soon as possible, which also helps reducing the switching delay.

The configuration discovery service can be invoked again *during* configuration discovery so that the client control point can add new front-ends to the discovery procedure (e.g., when it has discovered new front-ends after a handoff).

The configuration descriptions in the service's primitives are in terms of the configurations that a user can receive from his home aggregator (see the responsibilities of the front-end's control point, Section 3.4.2).

Requests

The control point on the mobile host invokes the configuration discovery service through a configuration discovery request primitive. The parameters of a request consist of a set of front-end URIs, an interface ID per URI, a channel name, the user's identity (e.g., bob@media-forward.nl) and credentials, and an optional description of a set of preferred configurations. The preferred configurations have to be a subset of the allowed configurations that appear in a user's delivery agreement with his home aggregator.

A discovery request also contains a parameter that specifies a maximum amount of time for configuration discovery, starting from the point at which the control point submits the request. This parameter ensures that the ALIVE client protocol entity discards late arrivals. When the configuration discovery service is re-invoked during discovery, the value of the maximum discovery time parameter is added to the total maximum discovery time.

Indications

An indication primitive signals that a switching controller is soliciting a description of the available configurations in which a user can receive a certain channel. An indication contains the identity of the user (e.g., bob@media-forward.nl), a channel name, and a description of a user's allowed configurations. If the request contained a set of preferred configurations, then the indication contains a description of that set of configurations. Notice that the control point of a front-end uses the identity of a user to identify the user's home aggregator (see Section 3.4.2).

An indication only occurs when a user can be authenticated. The configuration discovery service provider (specifically, the ALIVE protocol entities) transparently handles authentication, thus hiding the authentication process from the front-end's control point.

Responses

The control point of a front-end handles indications and returns the result in a response primitive. A response contains a channel name and a

description of the available configurations of that channel. The available configurations are a subset of the configurations in the indication.

A response also contains a status code that either indicates that the control point successfully served the request (OK), or that indicates that the aggregator cannot deliver a certain channel (NotFound). The latter typically happens when the aggregator does not have a forwarding agreement (see Section 2.2.5) with the source from which the channel originates.

Confirmations

Each response results in a discovery confirmation at the switching controller that issued the request. A confirmation primitive contains the configuration description of the corresponding response, the response's status code, and the URI of the front-end that issued the response. If the user could not be authenticated at the front-end, then the status code is Forbidden.

Summary

Table 3-1 summarizes the parameters of the configuration discovery service's primitives.

Table 3-1. Service primitives of the configuration discovery service.

Primitive	Parameters	Location
Request	Maximum discovery time, channel name, user ID, credentials, front-end URIs, interface for each front-end, preferred configurations (optional)	SC
Indication	Channel name, user ID, allowed configurations	FE
Response	Status code, channel name, description of available configurations	FE
Confirm	Front-end URI, status code, channel name, description of available configurations	SC

3.4.5 Capability Discovery Service

The capability discovery service enables the control point on the mobile host to discover the capabilities of front-ends (e.g., if they can notify the control point of changes in a user's available configurations). The behavior of the capability discovery service is the same as that of the configuration discovery service (see *Figure 3-14*), except that the primitives carry different parameters.

A capability discovery request contains a set of front-end URIs, an interface identifier per URI, and a maximum discovery period. An indication contains does not contain any parameters. A response primitive contains a description of the front-ends capabilities. The confirmation contains the URI of a front-end and the capability description returned by the corresponding front-end. Notice that a capability discovery request does not contain the user's credentials because the service does not require users to be authenticated.

Table 3-2 summarizes the primitives of the capability discovery service.

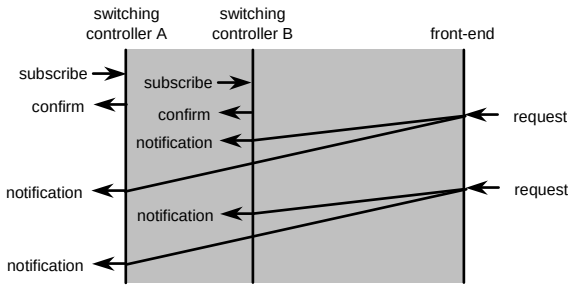
Table 3-2. Service primitives of the capability discovery service.

Primitive	Parameters	Location
Request	front-end URIs, interface for each front-end, maximum discovery time	SC
Indication	-	FE
Response	Description of capabilities	FE
Confirm	Front-end URI, description of capabilities	SC

3.4.6 Configuration Notification Service

The configuration notification service enables the client control point to detect changes in the availability of configurations. *Figure 3-14* shows the service's basic behavior, which involves a subscribe primitives, confirmation primitives, requests primitives, and notification primitives. Client control points use the subscribe primitive to subscribe to configuration notifications for certain front-ends. The configuration notification service manages the subscription of users to events, which means that a front-end control point does not receive an indication primitive when a user subscribes.

Figure 3-14. Behavior of the configuration notification service.



Subscribes

The switching controller has to subscribe to configuration notifications through a subscribe primitive. A subscribe primitive contains the user's identity and credentials, a URI of a front-end and the name of the interface through which it can be reached, a channel name, and an optional description of a set of configurations in which the switching controller is particularly interested.

Confirmations

A confirmation acknowledges the subscription of a user to the configuration notification service. It contains a status code that indicates if the subscription succeeded (OK), the URI of the front-end, and the channel name that also appeared in the subscribe request. The status code may also indicate that the front-end does not support configuration notifications

(NotSupported) or that the switching controller is not authorized to subscribe to the events (Unauthorized).

Requests

The control point at the front-end issues request primitives to notify subscribed switching controllers of changes in the aggregator's available configurations (see step 2 in Section 3.4.2). Each request primitive results in notifications at multiple switching controllers.

The request's parameters are the name of a channel, a set of aggregator IDs (the aggregators that appear in the front-end's roaming directory, see Section 3.4.2), and a description of available configurations per aggregator ID.

Notifications

A notification contains the URI of an aggregator's front-end, a channel name, and a description of the available configurations in which a user can currently receive the channel from the aggregator.

Provider-initiated Notifications

If a front-end does not support notifications, then the configuration notification service generates notifications on its own initiative, as shown in *Figure 3-15*. Provider notifications do not require a client control point to first issue a subscribe primitive. The parameters of a provider notification are the same as those of a notification to which the client control point explicitly subscribed.

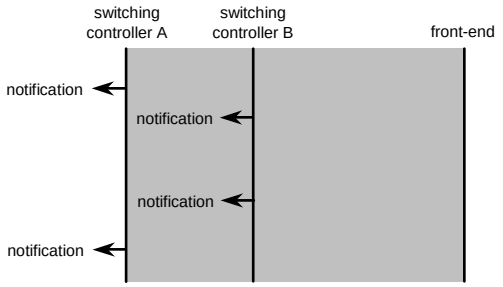


Figure 3-15. Provider notifications.

Summary

Table 3-3 summarizes the parameters of the notification service.

Table 3-3. Service primitives of the configuration notification service.

Primitive	Parameters	Location
Subscribe	Front-end URIs, interface for each front-end, channel name, user ID, credentials, maximum time to subscribe	SC
Confirmation	Status code, front-end URI, channel name	SC

Request	Channel name, aggregator IDs, description of available configurations per aggregator ID	FE
Notification	Front-end URI, channel name, description of available configurations	SC

3.4.7 Switching Service

The switching service enables the control point on the mobile host to switch the host to a target media server of the target aggregator. *Figure 3-16* shows the service's behavior (break-before-make strategy), which involves four types of primitives: requests, indications, responses, and confirmations.

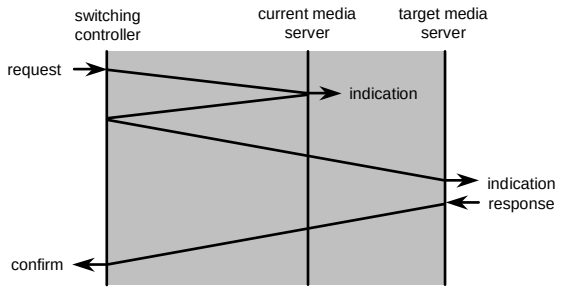


Figure 3-16. Behavior of the switching service.

Requests

The control point on the mobile host uses a switch request primitive to initiate a switch. A switching request contains the user's ID, his credentials, a description of a target configuration (the configuration the control point considers the best one) and two URIs, one of the current media server and one of the target aggregator. Each of the URIs comes with the identity of a network interface through which the media servers are can be reached. The request primitive furthermore contains a switching strategy (e.g., break-before-make or make-before-break), and the maximum amount of time that the ALIVE protocol entities can use to execute the switch. The final parameter is a session release delay, which has to be used in combination with a make-before-make switching strategy. The session release delay specifies how long the mobile host should continue to receive a channel from the current media server, starting from the point at which the mobile host begins to receive the channel from the target media server. The session release delay enables the mobile host to receive two copies of the same channel for a specified amount of time, which can help to keep the host's playout buffer (see Section 3.2.1) filled.

Indications

As a result of a request, a switching indication occurs at the target media server and optionally at the current media server. The order in which these

indications occur depends on the switching strategy. In a make-before-break switch, the indication will first occur at the target media server and then at the current media server. For break-before-make switches it will be the other way around.

The indication at the old media server is optional because a mobile host may no longer be able to reach its current media server, for instance because it roamed into a network where the current aggregator is no longer available. To deal with such situations, the control points of media servers have to be able detect mobile hosts that are no longer receiving streams from the media server (see Section 3.4.3).

A switch indication contains a user ID, a channel name, and a configuration description, all three of which are the same as in the switching request. The user's ID enables the control point on the media server to map the configuration description in the indication to a description of one of its own supported configurations (see media server control point, Section 3.4.3).

A switch indication only occurs if a user has been successfully authenticated. The ALIVE protocol entity on the media server checks this in a way transparent for the server's control point.

Responses

The control point on the target media server reacts to a switch indication with a switch response. The response contains a status code that indicates if the control point is willing to serve the switching controller.

The control point on the current media server does not issue a response.

Confirmations

A switch confirmation primitive signals that the switch has been executed (i.e., that the session with the old media server has been released and the session with the target media server has been established). A switch confirmation primitive contains the URI of the target media server and the status code that the target media server used in its response.

Summary

Table 3-4 summarizes the parameters of the switching service.

Table 3-4. Service primitives of the switching service.

Primitive	Parameters	Location
Request	Channel name, description of target configuration, user ID, credentials, current media server URI, target media server URI, interface ID per media server, switching strategy, maximum switching time, session release delay	SC
Indication	Channel name, configuration description, user ID	MS
Response	Status code	Target MS

Confirm	Status code, front-end URI	SC

3.4.8 Connectivity Handler Services

A connectivity handler provides an aggregator discovery, an aggregator notification, and a handoff service. The service primitives described in this section define the interactions between the client control point and the connectivity handler. The remote primitives are outside the scope of this thesis.

Aggregator Discovery Service

The aggregator discovery service enables the client control point to discover the local aggregators available on a certain network. The service's request primitive contains the name of one of the mobile host's interfaces and the name of a network on that interface. The confirmation that follows contains a set of URIs that point to the front-ends of the local aggregators on the network.

Aggregator Notification Service

The aggregator notification service enables the client control point to detect changes in the availability of local aggregators on a certain network. The service's primitives consists of a subscribe primitive (to subscribe to the service) and of notification primitives (to notify the control point of changes). The subscribe primitive contains the name of one of the host's interfaces and the name of a network on that interface. The parameters of a notification consist of an interface name, a network name, and a set of URIs that indicate which aggregators are currently available on the network.

Handoff Service

The handoff service enables the control point to execute a handoff on one of the mobile host's interfaces. The parameters of the request consist of an interface name, the identity of the current and target networks, the user's identity and credentials, and a handoff specification. The latter could for instance specify the maximum amount of time that the connectivity handler is allowed to spend on a particular handoff, or the strategy that it should use in executing the handoff (e.g., make-before-break).

The realization of the handoff service typically requires the connectivity handler to perform a considerable number of tasks, such as:

- Authenticate the user on the target network (e.g., using 802.1x for 802.11 networks [Mishra04, Pack02, Gast02]);
- Obtain an IP address for the interface (e.g., using DHCP [Droms99, Vatn98]);

- Perform a location update (e.g., using Mobile IP [Solomon98], SIP [Wedlund99, Kwon02], SLM [Landfeldt99], and so on); and
- Update the host's settings after a location update (e.g., adapting its routing tables [Peddemors04]); and
- Actually execute the handoff (e.g., between two 802.11 access points [Mishra03, Vatn03, Velayos03, DeCleyn04]).

The details of the execution of a handoff are however outside the scope of this thesis.

Summary

Table 3-5 summarizes the local service primitives of the connectivity handler.

Table 3-5. Service primitives of the connectivity handler.

Service	Primitives	Parameters
Aggregator discovery	Request	Interface, network
	Confirmation	Interface, network, set of front-end URIs
Aggregator notification	Subscribe	Interface, network
	Confirmation	Interface, network
	Notification	Interface, network, set of front-end URIs
Handoff	Request	Interface, current network, target network, user ID, credentials, handoff specification
	Confirmation	Status code

3.4.9 Server Monitor

A server monitor enables the control point of a front-end to check in which configurations the aggregator's media servers can currently stream out a particular channel. *Table 3-6* shows the server monitor's service. The configuration descriptions are in terms of the aggregator's own supported configurations.

Table 3-6. Service primitives of the server monitor.

Primitive	Parameters
Request	Channel name, description of supported configurations (optional)
Confirmation	Channel name, description of supported configurations plus URIs per configuration
Subscribe	Channel name
Notification	Channel name, description of configurations

The request primitive of the server monitor service contains the name of a channel and the description of one or more supported configurations (optional). A request results in a confirmation primitive that contains a description of the configurations in which the channel can currently be streamed off an aggregator's media servers, including a set of media server URIs per configuration. The configurations in the confirmation are a subset of those in the request, if any.

The server monitor service also contains a subscribe primitive with which the control point can subscribe to events that signal changes in the configurations in which media servers can deliver a certain channel (e.g., a change in the set of servers that can deliver a certain configuration). After calling the subscribe primitive, the server monitor server will issue notification primitives to indicate such changes.

Realization

Conceptually, the server monitor builds on a database that contains entries of the form <channel name, configuration descriptions, URIs per configuration description>. To serve requests, the server monitor matches the channel name and configuration descriptions in a request primitive with the information in the database, and returns the result in a confirmation primitive.

To detect changes in the availability of configurations, the server monitor has to be able to communicate with the media servers of an aggregator (e.g., by polling the media servers for their available resources such as processing load). The actual synchronization mechanism as well as the details of the server monitor's internal operation are however outside the scope of this thesis.

In general, the server monitor can be realized on a centralized server, or it can be distributed across the media servers of an aggregator [Amir98].

3.5 ALIVE Policies

One of the requirements of the ALIVE system is that it should enable users and other stakeholders (e.g., the system administrator of a set of mobile hosts or their manufacturers) to flexibly change the rules based on which the ALIVE system makes decisions (see Section 3.1.3). In this thesis, we use *policies* for this purpose [Kamilova05, Zhuang03]. In general, policies are rules with conditions and actions that are used by a controlling entity to continuously govern the behavior of a controlled entity. Policies have a goal, which means that the controlling entity aligns the behavior of the entire system (controlling entity plus controlled entity) with the goals of the policies.

The controlling entity is usually referred to as the Policy Decision Point (PDP), while the controlled entity is called the Policy Enforcement Point (PEP) [Westerinen01].

In a policy-based system, policies can typically be downloaded into a PDP from a (central) repository, thus changing the behavior of the entire system (i.e., PDP plus PEP) without halting it ('always on'). Another

advantage is that this enables policies to be enforced consistently across multiple devices (e.g., across the mobile hosts of a user).

The PDPs in the ALIVE system are part of the client control points and the control points of front-ends and media servers. The remaining parts of the control points plus the ALIVE protocol entities and the connectivity handler are PEPs.

In this section, we concentrate on the policies of the client control point. We first discuss which policies it can use (Section 3.5.1) and then consider a scenario in which policies control a switch (Section 3.5.2).

3.5.1 ALIVE Client Policies

ALIVE policies consist of the usual condition and action parts [Westerinen01] augmented with a clause that specifies the goal of the policy [Cox99]. The condition indicates when the policy fires (e.g., "if signal-to-noise ration greater than 4 dB and speed smaller than 20 km/hr"), the action indicates which service to invoke (e.g., "initiate configuration discovery") when the policy fires, and the goal represents what the policy is trying to accomplish (e.g., "smooth switching"). The ALIVE protocol entity and the connectivity handler must execute the actions of a policy when it fires, which means that the ALIVE policies are obligation policies [Cox99]. Other policy types (e.g., prohibition or authorization policies) are outside the scope of this thesis.

We distinguish two types of policies that a client control point can use: configuration discovery policies (when to initiate configuration discovery) and switching policies (when and how to initiate a switch). *Figure 3-17* shows an example of two configuration discovery policies (taken from [Kamilova05]). The two policies have different goals. The goal of the first policy is facilitate smooth switching (i.e., before the host's playout buffer empties), while the goal of the second policy is to do the same in a moderately smooth manner (e.g., allowing the buffer to be empty for some time). The policies realize their goals through different levels of proactivity. The first policy initiates configuration discovery when the number of lost multimedia packets on the mobile host's 802.11 interface increases to above 20%, while the second one waits until this is 50%. With the second policy, there is a higher chance of the playout buffer on the mobile host depleting, but it makes the host stick with the 802.11 for a longer period of time. This strategy might be preferable for 802.11 networks because they are typically cheaper to use than a network like UMTS.

Figure 3-17. Examples of configuration discovery policies.

```
<policy>CONFIGURATION_DISCOVERY
    <goal>
        <smoothness>HIGH</smoothness>
    </goal>
    <condition>
        <receiving_interface>WLAN</receiving_interface>
        <packet_loss>20
            <operator>GREATER_THAN</operator>
        </packet_loss>
    </condition>
    <action>
        <max_discovery_time>SHORT</max_discovery_time >
        <discover_alternatives/>
    </action>
</policy>
<policy>CONFIGURATION_DISCOVERY
    <goal>
        <smoothness>MODERATE</smoothness>
    </goal>
    <condition>
        <receiving_interface>WLAN</receiving_interface>
        <packet_loss>50
            <operator>GREATER_THAN</operator>
        </packet_loss>
    </condition>
    <action>
        < max_discovery_time>DEFAULT</ max_discovery_time>
        <discover_alternatives/>
    </action>
</policy>
```

The PDPs in the ALIVE system also use the goals of policies as a key to retrieve the proper policies from a policy repository, which can for instance contain policies described in XML (see the example of *Figure 3-17*). The client control point could for instance retrieve those policies from the repository that match the preferences of the user.

3.5.2 Operation

Figure 3-18 shows a scenario (taken from [Kamilova05]) in which the client control point (the PDP) makes a policy decision at point B in *Figure 3-1*. The environment monitor represents the local components on the mobile host (e.g., the local resource manager and the host's interfaces).

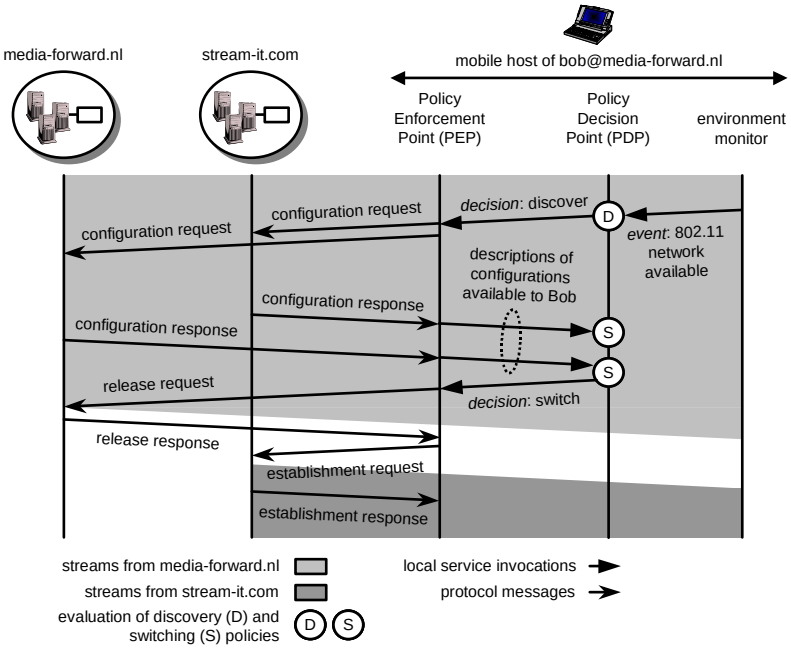


Figure 3-18. Policy-controlled scenario.

At point B, the client control point receives an event from the mobile host's 802.11 network interface indicating that a new 802.11 network has come into range (the network of hotspot.nl, see *Figure 3-1*). The event triggers the evaluation of the discovery policies in the client control point. In the example of *Figure 3-18*, this results in a configuration decision that indicates that the client control point should invoke the configuration discovery service (see Section 3.4.4). As a result of the service invocation, the mobile host exchanges configuration request and response messages with the front-ends of the two aggregators. The configuration responses carry descriptions of available configurations, in this case a description of the configurations in which Bob can receive CNN TV. The configuration request and response messages are part of the ALIVE protocol and will be discussed in detail in Section 3.6.

The client control point receives the descriptions of the available configurations in confirmation primitives (see Section 3.4.4). Each time the client control point receives a confirmation, it evaluates its switching policies. In the example of *Figure 3-18*, these policies make the client control point decide to switch Bob's mobile host to stream-it.com and perform the switch in a break-before-make manner (e.g., as part of a moderately smooth switching strategy). The establishment and release messages shown in *Figure 3-18* are part of the ALIVE protocol and will be discussed in Section 3.6 as well.

3.6 ALIVE Protocol

The ALIVE protocol realizes the ALIVE services of Section 3.4. The ALIVE protocol consists of the messages exchanged between switching controllers and front-ends and between switching controllers and media servers. These interactions take place on the signaling associations of Section 3.2.4. The AAA interactions between front-ends are not part of the ALIVE protocol.

In this section, we concentrate on the ALIVE-specific part of the protocol. We assume that more common protocol functions are in place. In particular, we assume that the ALIVE protocol entities provide some sort of transaction management, that they take care of addressing, and that messages are delivered reliably. An implementation of the ALIVE protocol (see Section 3.7) needs to realize these functions.

3.6.1 Message Overview

Table 3-7 provides an overview of the ALIVE protocol messages, specifically for which services they are used and what their purpose is. Each message type consists of a request and a response (e.g., a capability request and a capability response).

Table 3-7. Overview of ALIVE protocol messages.

Service	Messages	Purpose
Configuration discovery	Authentication request & response	Authenticate a user at a front-end and cache his authentication state
	Configuration request & response	Get a description of the available configurations in which a user can receive a channel from a front-end
Configuration notification	Subscribe request & response	Subscribe to configuration notifications
	Configuration notification	Notify switching controllers of changes in the set of available configurations in which a user can receive a channel
Refresh authentication state (hidden from control points)	Refresh request & response	Refresh a user's authentication state at a front-end
Capability discovery	Capability request & response	Get a description of a front-end's capabilities
Switching	Establishment request & response	Establish a multimedia session with a media server
	Release request & response	Release a multimedia session with a media server

3.6.2 Client-side Protocol Entity

The ALIVE client protocol entity maintains a block of *state* for each front-end with which it has successfully authenticated the user. This enables it to send each front-end a regular refresh request to keep the user's

authentication state alive (see Section 3.3.1). The per-front-end state also enables the ALIVE protocol entity to store the properties of front-ends (e.g., the refresh intervals that they accept and the authentication tokens they issued) and hide that information from the control point. This simplifies the interactions between the ALIVE protocol entity and the control point. For example, the control point only has to inform the ALIVE protocol entity of new front-ends and not of already discovered front-ends.

Configuration Discovery

When the ALIVE client protocol entity receives a configuration discovery request primitive, it sends configuration requests to known front-ends (i.e., to front-ends in the request primitive for which it maintains state) and authentication requests to new front-end (i.e., to front-ends in the request primitive for which it does not yet maintain state).

A *configuration request* contains a channel name, an authentication token (see below), and an optional set of preferred configurations. The client protocol entity copies the channel name, the token, and the preferred configurations from the front-end's state. The state also indicates through which interface the protocol entity should transmit the request.

The front-end react to the configuration request with a *configuration response*, which contains a status code, a channel name, and a description of the available configurations of a channel (including media server URIs, see Section 3.3.2). The status code either indicates that the request was successfully served (`OK`) or that the aggregator cannot deliver the channel (status code `NotFound`). The latter situation usually occurs because the aggregator does not have a forwarding agreement with the source of the channel (see Section 2.2.5). The client protocol entity returns the URI of the front-end that sent the response, the status code, and the configuration description to the client control point in a confirmation primitive.

An *authentication request* contains the user's ID, his credentials, and a proposal for an authentication refresh interval (see Section 3.3.1). The client protocol entity copies the user's ID and his credentials from the request primitive and uses a default value for the refresh interval. The configuration discovery request primitive also indicates through which interface the client protocol entity has to transmit the request.

A front-end reacts to an authentication request with an *authentication response*. The response contains a status code that indicates the outcome of the authentication request. The status code is OK if the front-end successfully authenticated the user and accepted the refresh interval in the request. In this case, the authentication response also contains an authentication token (see Section 3.3.1) that the ALIVE client protocol entity must include in any further requests it sends to the front-end. The ALIVE client protocol stores the authentication token and other

information about the front-end (e.g., the refresh interval and the interface through which it can be reached) in a block of state that represents the front-end and immediately sends a *configuration request* to the front-end (see above).

The status code of an authentication request can also indicate that the front-end did not accept the refresh interval in the request (status code IntervalUnacceptable). In this case, the client protocol entity has to resubmit the authentication request. To ensure that the client protocol selects an acceptable refresh interval, the authentication response contains the range of refresh intervals that the front-end does find acceptable.

The status code of an authentication response can furthermore indicate that the authentication request did not contain the user's credentials (Unauthorized), in which case the client protocol entity has to resubmit the request with the user's credentials. Another possibility is that the authentication response indicates that the front-end could not authenticate the user (status code Forbidden). In this case, the client protocol entity issues a configuration discovery confirm primitive with the same status code. This typically occurs when the aggregator does not have roaming agreement with the user's home aggregator.

The ALIVE client protocol entity uses a configuration discovery timer to cap the discovery process at the maximum value specified in a configuration discovery request primitive. When the client control point reinvokes the configuration discovery service during discovery, then it adds the value in the new request primitive to the current value of the configuration discovery timer.

The discovery procedure ends when either all the front-ends that received a configuration request have reacted with a configuration response, or when the discovery timer expires.

Capability Discovery

Upon receiving a capability discovery request primitive, the client protocol entity transmits a *capability request* message to each of the front-ends in the request primitive. The request primitive also specifies through which the requests should be sent.

Each front-end replies with a *capability response*, which contains a description of the front-end's capabilities (see Section 3.4.5). The client protocol entity passes this information to the local control point in a capability discovery confirmation primitive, which also contains the URI of the front-end that sent the message. Notice that front-ends also server capability requests from unauthenticated users.

Capability discovery ends when at the end of the maximum discovery time specified in the request primitive, or when every front-end has returned a capability response message.

Configuration Notification

When the ALIVE client protocol entity receives a subscribe request primitive from the client control point, it sends a *subscribe request* message to known front-ends (i.e., front-ends for which the client protocol entity maintains state) and an authentication request to new front-ends (i.e., for which the protocol entity does not yet maintain state).

A *subscribe request* message contains a token and a channel name. The client protocol entity copies the channel name from the subscribe request primitive and the token from the front-end state that a client protocol entity maintains. The front-end state also indicates through which interfaces the request should be sent.

The front-end returns a *subscribe response* message that indicates if the subscribe succeeded. The client protocol entity passes the result to the client control point in a confirmation primitive. If the subscribe request fails, then this is typically because the front-end does support eventing (NotSupported status code), or because the request contained an invalid token (Unauthorized status code).

The client protocol entity constructs an authentication request using the information in the subscribe request service primitive (interface name, user ID and credentials) and a default authentication refresh interval. The authentication procedure is the same as during configuration discovery (see above), except that an authentication response with an OK status code is followed by the transmission of a subscribe request message instead of a configuration request message. If the client protocol entity receives an authentication response with a Forbidden status code, then it pass this code to the client control point using a subscribe confirmation primitive.

If the subscribe request was successful, the client protocol entity merely needs to wait for *configuration notification* messages from the front-end. A configuration notification message contains the name of the channel and a description of the available configurations in which the user can currently receive the channel from an aggregator. The client protocol entity passes this information plus the URI of the front-end that sent the message to the client control point in a configuration notification primitive.

For provider-initiated notifications, the ALIVE client protocol entity regularly sends configuration requests to that front-end to get a description of the user's current set of available configurations (e.g., together with a refresh request). To detect changes, the ALIVE protocol entity could add the most recent description of available configurations that it received to the front-end's state (e.g., as a hash). It can then locally compare the configuration descriptions in the next configuration response with the stored version and pass a configuration notification primitive to the control point if there is a difference.

A subscribe request also contains the address of the switching controller, which enables it to also use a subscribe message to update its location at a front-end, typically after a handoff.

Switching

When the ALIVE client protocol entity receives a switch request primitive from the control point, it first checks how it needs to switch the mobile host to the target media server. If the switching strategy is break-before-make, then the switching controller will first release the multimedia session with the target media server by sending a *release request* to it. If it is make-before-break, then it will first send an *establishment request* to the target media server. The client protocol entity uses the URIs in the switch request primitive to transmit these messages.

An establishment request contains the user's token for the target aggregator (obtained from the front-end's state) and a description of the intended actual configuration. A release request contains the user's token for the current aggregator and a description of the actual configuration in which the mobile host was receiving the channel (both obtained from the front-end state).

Media servers reply to an establishment request with an *establishment response*. Similarly, they use a *release response* to answer a release request. Both reponse messages carry a status code that indicates if the media server successfully executed the request.

A switch ends when the client protocol entity receives an establishment response (break-before-make) or when it receives a release response (make-before-break). The client protocol entity passes the status code in the establishment response to the client control point in a confirmation primitive. The status code will be negative if the client protocol entity could not finish the switch in time.

If the switching strategy is make-before-break and the client protocol entity receives an establishment response, then it waits until the end of the release delay before sending a release request message to the current media server.

Refreshing Authentication State

The client protocol entity uses a per-front-end refresh timer to refresh the user's authentication state at a front-end. The value of the timer for a particular front-end is determined during the authentication request-response with that front-end (see Configuration Discovery).

When the refresh timer expires, the client protocol entity transmits a *refresh request* to the front-end. The refresh request carries the user's token for the front-end.

The front-end replies with a *refresh response*, which indicates if the front-end successfully refreshed the user's authentication state (OK). If this is not the case, the switching controller either did not include a token, or it submitted the refresh request too late. The status code signals both cases through the same status code (Unauthorized).

The ALIVE client protocol entity restarts the refresh timer for a particular front-end if it receives a positive refresh response from that front-end.

3.6.3 Front-end Protocol Entity

The ALIVE front-end protocol entity is the peer of the ALIVE client protocol entity. It maintains a block of state for each switching controller whose user it successfully authenticated. Each block contains information like the channel the mobile host is receiving and in which configuration, the authentication token, the refresh interval that the ALIVE client protocol entity uses, the user's set of allowed (foreign) configurations, if the switching controller subscribed to configuration notifications, and so on. An ALIVE protocol entity on a mobile host keeps its block at a front-end of state alive through refresh request messages. The front-end protocol entity deletes this state if the mobile-side ALIVE entity does not provide a refresh request in time.

Figure 3-19 shows that the front-end protocol entity contains a AAA bridge that interfaces with the front-end's AAA server. The AAA bridge enables the front-end protocol entity to invoke the services of a AAA server, which is a local service to authenticate a user and to get a description of a user's allowed (foreign) configurations. The AAA server also generates the user's authentication token (see Section 3.3.1). We refer to Section 3.6.5 for the AAA server's interface specification.

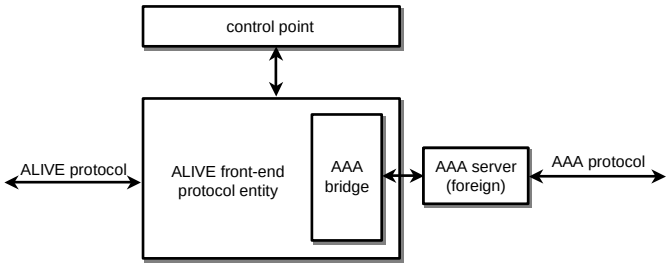


Figure 3-19. Internal organization of the front-end protocol entity.

Configuration Discovery

When the ALIVE front-end protocol entity receives an *authentication request*, it first checks if the request includes the user's credentials and if the refresh

interval in the request is acceptable. If this is the case, the protocol entity's AAA bridge invokes the services of the AAA server to authenticate the user.

When the AAA server successfully authenticates the user, the front-end protocol entity creates a block of state that represents the user's switching controller. The front-end protocol entity adds a description of the user's allowed configurations and an authentication token to the block of state, thus essentially forming an *authentication cache* (see Section 3.3.1). The front-end protocol entity also adds the switching controller's refresh interval to the block of state. Next, the protocol entity returns an *authentication response* to the client protocol entity, which includes the user's token and a status code.

The front-end protocol entity returns an authentication response with a Forbidden status code if the AAA server could not authenticate the user. If the refresh interval in the authentication request is unacceptable, then it returns an authentication response with a status code of IntervalUnacceptable. In this case, the authentication response also contains the range of refresh intervals that the front-end does find acceptable. If the request does not contain the user's credentials, the protocol entity returns an Unauthorized response.

If the front-end protocol entity receives a *configuration request*, it first checks if the token in the request matches the token in the switching controller's block of state. If this is the case, the protocol entity passes an indication primitive to the front-end's control point. The indication contains a description of a set of allowed configurations, specifically the intersection of the allowed configurations in the request (preferred configurations) and the allowed configurations that appear in the user's block of state. The protocol entity also includes the user's ID in the indication and the channel name from the configuration request message.

When the protocol entity receives a configuration discovery response primitive, it constructs a *configuration response* messages and includes the description of the available configurations in the response primitive (which includes media server URIs) in the response message. The protocol entity also copies the status code in the response primitive into the configuration response message. If the control point indicates that it cannot deliver the requested channel, the status code is NotFound.

The protocol entity does not generate an indication primitive if the configuration request includes an invalid token. In this case, the protocol entity returns a configuration response with an appropriate status code (Unauthorized).

Capability Discovery

If the front-end protocol entity receives a capability request, it generates a capability discovery indication primitive. When the local control point answers with a response primitive, the protocol entity copies the capability

description in the response primitive to a capability response message and sends it to the switching controller that sent the capability request message.

Configuration Notification

When the front-end protocol entity receives a subscribe request message from a switching controller, it first checks if the token matches. If this is the case, the protocol entity adds the channel name in the subscribe message as well as the set of preferred configurations in the message to the user's block of state. Next, it returns a subscribe response with status code OK. If the token in the request does not match, the protocol entity returns an Unauthorized status code in the subscribe response. If the front-end does not support configuration notifications, it immediately returns a subscribe response with a NotSupported status code.

When the front-end protocol entity receives a configuration notification request from the local control point, it checks which switching controllers have subscribed to receive notifications about the channel specified in the request and sends a configuration notification message to each of them. The configuration description in a particular notification messages consists of the intersection of the available configurations in the request primitive that match the ID of the user's home aggregator (see *Table 3-3*) and the user's allowed configurations (in the switching controller's state).

If the front-end protocol receive a subscribe message with a new IP address (typically after a handoff), then it updates the state of the corresponding switching controller with the new address.

Refreshing Authentication State

The front-end protocol entity maintains a refresh timer for each authenticated switching controller. When the front-end protocol entity receives a refresh request from a switching controller, it checks if the token matches and restarts the refresh time for that user. It then transmits a refresh response that indicates that the state was successfully refreshed.

If the refresh timer times out, the front-end protocol entity deletes the state associated with the user.

3.6.4 Media Server Protocol Entity

When the ALIVE protocol entity on the media server receives an establishment request, it first checks with the front-end if the token is valid. If this is the case, it generates an indication primitive. The protocol entity copies the parameters in the indication from the establishment request (channel name, configuration description, and user ID).

When the ALIVE protocol entity receives an establishment response primitive from the local control point, it returns the contents of the

primitive (a status code) to the switching controller in an establishment response message.

Release requests are handled in the same way, except that the protocol entity immediately transmits a release response after it issued the release indication primitive.

3.6.5 AAA Server

A AAA server provides a combined authentication and authorization service. The service primitives described in this section define the interactions between the ALIVE media server protocol entity and the AAA server. *Table 3-8* shows the service's primitives.

Table 3-8. Service primitives of the AAA server.

Primitive	Parameters
Request	User ID, credentials, description of preferred configurations (optional)
Confirmation	Success/failure, token, description of allowed (foreign) configurations (a subset of the preferred configurations, if any)

The request primitive contains the user's identity (e.g. bob@media-forward.nl), the user's credentials, and the description of a set of preferred configurations (optional). The resulting confirmation contains the outcome of the authentication/authorization (success or failure). If the outcome is positive, the confirmation contains a token and a description of the user's allowed (foreign) configurations.

If an authentication request fails, then this is usually because a user attempts to authenticate with a foreign aggregator that does not have a roaming agreement with the user's home aggregator.

Realization

The authentication and authorization service mainly builds on *user directories*. Each front-end has one user directory, which contains an entry for each user that has a subscription with the aggregator. Each entry consists of a user identity (e.g., bob@media-forward.nl) and a description of the user's allowed configurations, which are part of the delivery agreement between the user and the aggregator (see Chapter 2). A user directory is similar to the home location register in cellular networks such as GSM and UMTS [Køien03].

When the AAA server receives a request primitive, it first checks if the user's identity appears in the local user directory. If this is the case, the AAA server authenticates the user locally. If the user is a foreign user (i.e., his identity does not appear in the user directory), then the AAA server delegates authentication to the AAA server of the user's home aggregator (the home AAA server). For example, stream-it.com's AAA server will attempt to authenticate Bob at his home aggregator (media-forward.nl)

because Bob does not appear in stream-it.com's user directory. The AAA server passes the result of the authentication to the ALIVE media server protocol entity in an authentication confirmation.

The authorization part consists of simply retrieving the description of the user's allowed configuration from the user directory. For foreign users, the AAA server gets a description of the user's allowed configurations from his home aggregator, typically as part of the authentication procedure.

The AAA server uses a confirmation primitive to return the result of the authentication and the description of the allowed (foreign) configurations to the ALIVE server protocol entity.

AAA servers typically interact with each other through AAA protocols such as Diameter [Calhoun03]. The details of these interactions are however outside the scope of this work. The same goes for the internal operation of AAA servers (e.g., in terms of authentication mechanisms).

3.6.6 Message Summary

Table 3-9 summarizes the messages of the ALIVE protocol, their direction, and the information they carry. SC stands for Switching Controller, FE for Front-end, and MS for Media Server.

Table 3-9. ALIVE protocol messages, their direction, and the information they carry.

Message	Type	Direction	Data
Authentication	Request	SC → FE	User ID, password, refresh interval
	Response	FE → SC	Status code, token, refresh interval range
Configuration	Request	SC → FE	Channel name, token, description of preferred configurations (optional)
	Response	FE → SC	Status code, channel name, description of available configurations (including media server URIs)
	Notification	FE → SC	Channel name, set of available configurations (including media server URIs)
Subscribe	Request	SC → FE	Channel name, token, IP address
	Response	FE → SC	Status code
Capability	Request	SC → FE	-
	Response	FE → SC	Capability description
Refresh	Request	SC → FE	Token
	Response	FE → SC	Status code
Establishment	Request	SC → MS	Token, channel name, description of actual configuration
	Response	MS → SC	Status code
Release	Request	SC → MS	Token, channel name, description of actual configuration
	Response	MS → SC	Status code

3.6.7 Typical Scenario

Figure 3-20 shows a typical interaction of the ALIVE protocol using the example of *Figure 3-1*. The message exchange begins with when the client control point on Bob's mobile host decides to initiate configuration discovery as a result of Bob moving into the 802.11 network of hotspot.nl (see *Figure 3-1*, point B). When this happens, Bob is receiving CNN TV from his home aggregator media-forward.nl via the UMTS network of connect-it.nl.

The client control point initiates configuration discovery by calling the request primitive of the corresponding service. As a result, the client protocol entity transmits a configuration request to the front-end of media-forward.nl. At the same time (but shown after the configuration response from media-forward.nl for readability), the client control point uses the connectivity handler to connect Bob's mobile host to the 802.11 network of hotspot.nl. When the connection has been established, the client control point discovers local aggregator stream-it.com by calling the connectivity handler's aggregator discovery service. In *Figure 3-20*, we represented the resulting protocol invocation as an aggregator discovery request-response interaction (e.g., piggy-backed on DHCP) with hotspot.nl's aggregator directory.

Next, the client control point obtains a description of stream-it.com's capabilities by invoking the capability discovery service. As a result, the client protocol entity transmits a capability request to front-end of stream-it.com. The description in the capability response is such that the client control point decides to reinvoke the configuration discovery service to add stream-it.com to the discovery procedure. Since the client protocol entity does not maintain any state on stream-it.com, it sends an authentication request to that aggregator's front-end. However, the status code in the authentication response indicates that the client protocol entity proposed an unacceptable refresh interval in the request (IntervalUnacceptable). As a result, the client protocol entity resubmits the authentication request, this time with a refresh interval from stream-it.com's acceptable range of refresh intervals (in the authentication response). The authentication response following this request indicates that Bob was successfully authenticated through a AAA interaction with Bob's home aggregator.

After that, the client protocol entity sends a configuration request to stream-it.com's front-end. The configuration response contains a description of the available configurations in which Bob can receive CNN TV from stream-it.com. The available configurations are described in terms of Bob's allowed configurations, thus extending Bob's home environment at media-forward.nl to stream-it.com.

Based on the configuration response from stream-it.com, the client control point on Bob's mobile host decides that stream-it.com can deliver CNN TV in a better configuration. As a result, it switches the mobile host to one of stream-it.com's media servers by invoking the switching service. In this example, the client control point requests a break-before-make switch, which means that the client protocol entity first releases the multimedia session with the current media server at media-forward.nl and then establishes a session with a target media server of stream-it.com.

The switch completes when the client protocol entity receives an establishment response from the target media server. At that point, the mobile host is receiving CNN TV from one of stream-it.com's media servers via the 802.11 network of hotspot.nl.

Notice that the switching controller interacts with stream-it.com via the 802.11 network and with media-forward.nl via the UMTS network.

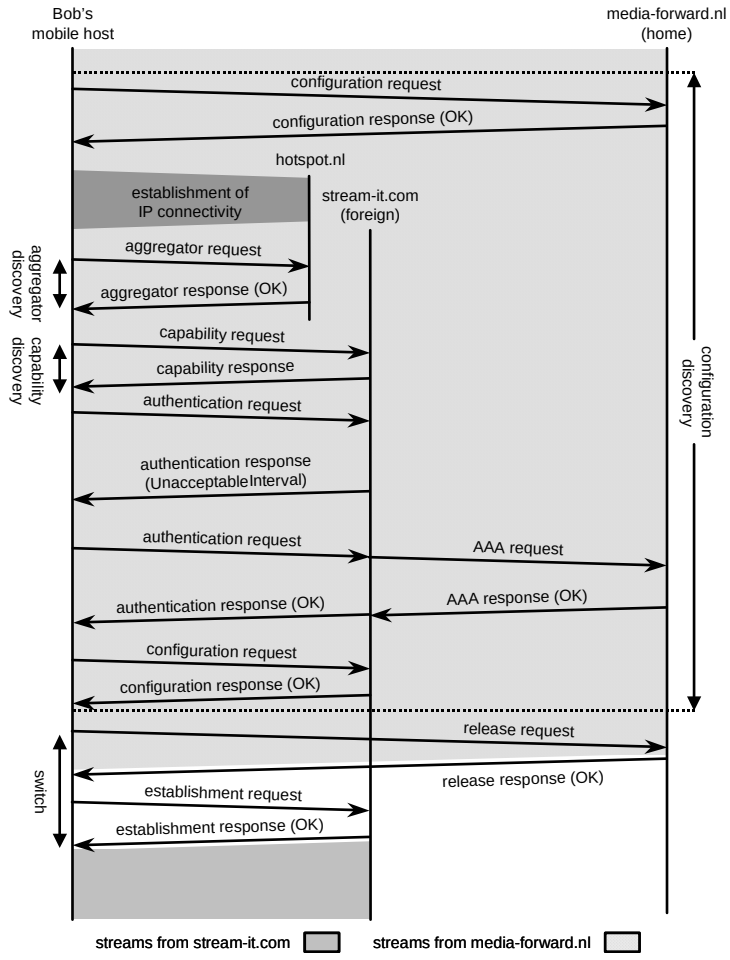


Figure 3-20. Typical interaction.

3.7 Implementation of the ALIVE Protocol

The ALIVE protocol of Section 3.6 requires one or more underlying transport protocols to convey the ALIVE protocol messages. The transport protocol between a switching controller and a front-end will typically be a single ubiquitously available protocol, while multiple transport protocols may be required to connect a switching controller to the media servers (e.g., SIP, RTSP, and WindowsMedia servers) of aggregators.

In this section, we consider an implementation of the ALIVE protocol that is based on a single transport protocol, specifically the Session Initiation Protocol (SIP) [Rosenberg02a]. We implemented the provider-

initiated notification service, but did not implement the configuration notification service that involves explicit notification from front-ends. We also did not implement the capability discovery service.

We first explain why we used SIP to implement the ALIVE protocol (Section 3.7.1) and briefly introduce the notion of a transaction, which is SIP's basic form of interaction (Section 3.7.2). After that, we explain the mapping of ALIVE messages to SIP messages (Section 3.7.3) and discuss the configuration discovery part of the ALIVE protocol (Section 3.7.4) as well as the part that executes a switch (Section 3.7.5). We conclude this section with the description of a small-scale testbed in which we deployed the implementation (Section 3.7.6).

3.7.1 Session Initiation Protocol

The Session Initiation Protocol (SIP) [Rosenberg02a] is an application-level signaling protocol for establishing, modifying, and tearing down multimedia sessions in the Internet. It uses textual messages, which typically carry a payload that describes multimedia sessions. SDP (see Section 3.3.2) is one of the languages that can be used for such descriptions.

We used SIP to realize the ALIVE protocol for the following reasons:

- SIP is typically used to convey descriptions of multimedia sessions (descriptions of available configurations, in our case), for instance to get a description of the other party's capabilities [Rosenberg02b];
- SIP can be run on top of UDP. The advantage of using UDP is that UDP messages can be sent right away without having to wait for a connection to be established (e.g., a TCP connection), which allows a switch to take place more quickly. The disadvantage of using UDP is that it limits the number of configuration descriptions that can be transferred to the maximum size of a UDP packet. In this thesis, we assume that the configuration descriptions that a mobile host receives from an aggregator fit into one UDP packet;
- SIP reliably transfers messages and provides transaction management (e.g., to be able to match different configuration responses with the corresponding requests);
- SIP can be used to refresh softstate (in our case the state maintained by ALIVE front-end protocol entities) [Donovan02];
- SIP provides hooks for shared secret user authentication;
- SIP's main purpose is to set up multimedia sessions, which enables mobile hosts to also use SIP to establish a multimedia session with a media server. This reduces the number of protocols in the system, thus reducing the system's complexity. We do however stress that aggregators can also use other session control protocols on their media servers (e.g., RTSP [Schulzrinne98] or WindowsMedia);

- It seems that SIP will eventually become a ubiquitously available protocol that is used for many purposes. SIP has for instance been adopted as the signaling protocol for UMTS multimedia sessions [Wong03];
- SIP is reasonably bandwidth efficient compared to the relatively high bandwidth levels that streaming applications usually require; and
- SIP uses textual messages, which are easier to extend, process, and debug than binary encoded messages.

Finally, SIP can also be used to extend the ALIVE protocol beyond its current capabilities. SIP eventing [Roach02] could for instance enables the control points of aggregators to push configuration notification messages to interested mobile hosts. SIP is currently also being extended with support for public key authentication [Peterson03].

3.7.2 SIP Transactions

SIP interactions are organized in so-called *transactions*. A transaction is a sequence of SIP messages that begins with a SIP client transmitting a *request* to a SIP server. The SIP server responds with zero or more so-called *provisional responses* followed by one *final response*. A provisional response informs the client that the server is handling the request, while a final response indicates that the server has executed the client's request. Depending on the type of request, clients confirm the receipt of a final response by transmitting an acknowledgement (an 'ACK') to the server. *Figure 3-21* shows this basic behavior.

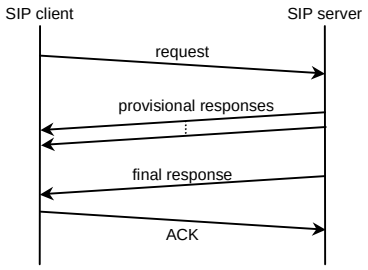


Figure 3-21. Basic SIP transaction.

Transaction Types

SIP supports two types of transactions: invite transactions and non-invite transactions. An *invite transaction* is a transaction that begins with an INVITE request, which is the request type that SIP for instance uses to establish a multimedia session or to requests another party's capabilities [Rosenberg02b]. A *non-invite transaction*, on the other hand, starts with a request messages that is not an INVITE. These for instance include

OPTION requests (another means to ask a SIP server for its capabilities), and SUBSCRIBE requests (to subscribe to event notifications) [Roach02]. In our implementation, we only used invite transactions.

A SIP client involved in an invite transaction issues an ACK when it receives a final response from a server. Non-invite transactions do not involve an ACK. Strictly speaking, an ACK is not part of an invite transaction if the final response is a 200 OK (see below), while an ACK is part of the transaction for other final responses.

Response Types

SIP supports different types of provisional and final responses. The type of a response is determined by its *status code*, which is an integer number between 100 and 699. A provisional response has a status code between 100 and 199, while the status code of a final response falls in the range 200-699.

The most important status code is 200 (OK). A response with this status code signals that the server has successfully executed the request of the client (e.g., an INVITE request). An example of a provisional status code is 100 (Trying), which informs the client that the server is trying to execute its request. All other status codes indicate the client needs to redirect its request to another SIP server (300-399), or that the SIP server could not serve the client's request (400-699).

Dialogs

SIP transactions take place in the context of a *dialog*, which is a signaling association between a SIP client and a SIP server. A dialog is identified by a tuple of three random numbers. Each SIP message contains the identifier of the SIP dialog to which it belongs in the form of three protocol headers (To tag, From tag, and Call ID, see RFC 3261 for details).

Transactions within a dialog are identified by the dialog's identifier and a sequence number. Every new transaction has a sequence number that is one higher than the transaction before it. Like the dialog identifier, every SIP message also carries the sequence number of the transaction (CSeq header).

Dialogs are established through INVITEs. The INVITE that triggers the establishment of a dialog is called an *initial INVITE*. INVITEs that are sent across an existing dialog are referred to as *re-INVITEs*.

3.7.3 Messages

Table 3-10 shows the mapping from ALIVE to SIP messages. To reduce the switching delay, we piggyback messages onto each other (e.g., a configuration request on an authentication request).

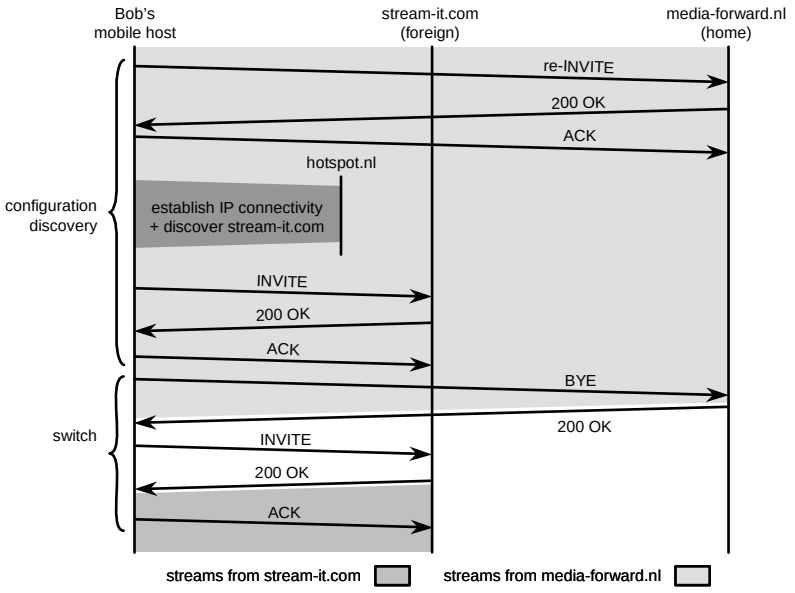
Table 3-10. Mapping of ALIVE messages to SIP messages.

ALIVE protocol message	Also carries	SIP message
Authentication request	Configuration request	INVITE
Authentication response	Configuration response	200 (OK), 401 (Unauthorized), 403 (Forbidden), 404 (NotFound), or 422 (IntervalTooShort)
Configuration request	Refresh request	Re-INVITE
Configuration response	Refresh response	200 (OK) or 401 (Unauthorized)
Refresh request	Configuration request	Re-INVITE
Refresh response	Configuration response	200 (OK) or 401 (Unauthorized)
Establishment request	-	INVITE
Establisment response	-	200 (OK) or 401 (Unauthorized)
Release request	-	BYE
Release response	-	200 (OK) or 401 (Unauthorized)

3.7.4 Configuration Discovery

Figure 3-22 shows the typical discovery behavior of the ALIVE protocol when the client control point invokes the configuration discovery service, in this case at point B of *Figure 3-1*.

Figure 3-22. Typical discovery behavior of the ALIVE protocol.



Observe that the switching controller on Bob's mobile host can interact with hotspot.nl and media-forward.nl (via UMTS) simultaneously. *Figure 3-22* only shows these interactions sequentially for readability. Also notice that *Figure 3-22* does not show the AAA interactions between stream-it.com

and media-forward.nl to authenticate Bob. We will discuss the switching part of *Figure 3-22* in Section 3.7.5.

INVITEs

An INVITE is a combined authentication-configuration request and also establishes a dialog with a front-end. The client protocol entity transmits an INVITE to front-ends with which it has not authenticated the user, in *Figure 3-22* to the front-end of stream-it.com (via the host's 802.11 interface).

Each INVITE contains the user's ID in the From header and the user's credentials in the Authorization header [Rosenberg02a]. An INVITE also contains SDP with the name of the channel the user wants to receive in the s= field. The SDP may contain a description of a set of allowed configurations (the preferred configurations).

The SDP is 'inactive', which means that the last line in the SDP payload is an a=inactive line. The a=inactive line informs an aggregator that the SDP describes capabilities rather than a particular configuration in which the aggregator should begin to stream the channel. This mechanism is similar to the SDP offer/answer model standardized by the IETF [Rosenberg02b]. However, unlike [Rosenberg02b], we use this mechanism in a multiparty fashion because a client protocol entity will generally query multiple aggregators at the same time.

An INVITE furthermore contains Session-Expires header [Donovan02], which the ALIVE protocol entity on the mobile host uses to propose an interval for refreshing the user's authentication state at a front-end (see Section 3.6.2).

Re-INVITEs

A re-INVITE is a combined configuration-refresh request. The client protocol entity sends a re-INVITE to a front-end to query it for its available configurations (i.e., as a configuration request) when it has already authenticated the user with that aggregator. In the scenario of *Figure 3-22*, the client protocol entity transmits the re-INVITE to media-forward.nl (via the host's UMTS interface). Since a re-INVITE also acts as a refresh message, the ALIVE front-end protocol entity refreshes a user's state when it receives a re-INVITE.

Re-INVITEs contain the same type of information as INVITEs, except that a user's credentials are replaced by an admission token. A re-INVITE also contains 'inactive' SDP.

The client protocol entity also regularly transmits re-INVITEs as refresh requests (cf. [Donovan02]). In this case, the re-INVITE also acts a configuration request. The length of the transmission interval is aggregator-specific. For simplicity, we have omitted the regular re-INVITEs from *Figure 3-22*.

200 OKs

A front-end protocol entity responds to an INVITE with a 200 OK if it has successfully authenticated the user and has accepted the proposed refresh interval in the INVITE. A 200 OK is thus a positive authentication-configuration response message. A 200 OK also establishes a SIP dialog between the switching controller and the front-end.

A 200 OK contains the refresh interval in the Min-SE header [Donovan02] and an admission token in the Admission-Authorization-Token header (cf. the P-Media-Authorization-Token of [Marshall03]).

The 200 OK's SDP is also 'inactive' and contains the name of the channel, a description of the user's potential configurations, and URIs to media servers. *Figure 3-23* shows an example for channel CNN Radio.

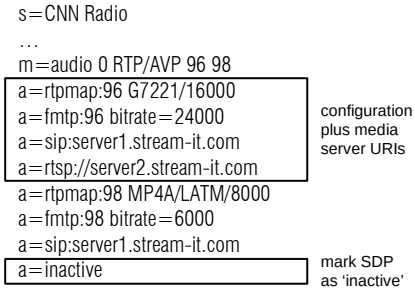


Figure 3-23. 'Inactive' SDP description of potential configurations.

Aggregators react to a re-INVITE with a 200 OK if they have successfully re-authenticated the user, either if the re-INVITE represents a configuration request (with a piggybacked refresh request) or if it represents a refresh request (with a piggybacked configuration request). These 200 OKs contain the same sort of information as a 200 OK to an INVITE.

The SDP in a 200 OK that is the result of regular re-INVITE (i.e., a refresh request with a piggy-backed configuration request) enables a client protocol entity to regularly detect changes in a user's set of potential configurations ('polling'). A more efficient and timely approach would be to use an announcement protocol [Roach02] (i.e., realize the eventing service of Section 3.4.5), but the use of such a protocol in the ALIVE system is an item of future work.

The request-response procedure of the ALIVE protocol is similar to that of SDP's offer/answer model [Rosenberg02b].

401, 403, 404, and 422 Responses

The ALIVE protocol uses four error responses (in general, SIP error responses lie in the range 300-699):

- 401 Unauthorized: the INVITE does not contain the user's credentials;
- 403 Forbidden: the aggregator cannot authenticate the user, typically because the aggregator does not have a roaming agreement with the user's home aggregator;
- 404 Not Found: the aggregator cannot deliver the channel for which the client protocol entity requests potential configurations (usually because the aggregator does not have an agreement with the source from which the channel originates); and
- 422 Session Interval Too Small: the aggregator requires a larger refresh interval.

Figure 3-24 illustrates that a 401 and a 422 response cause the client protocol entity to resubmit the request with other information. As a result, a client protocol entity might be involved in multiple SIP transactions before it receives a 200 OK. In the example of *Figure 3-24*, the first INVITE to stream-it.com does not contain the user's credentials. The aggregator responds to that INVITE with a 401 Unauthorized [Rosenberg02a] asking the client protocol entity to resubmit the INVITE with the user's credentials. The second INVITE in *Figure 3-24* contains these credentials, but stream-it.com considers the proposed refresh interval too small. Stream-it.com therefore returns a 422 Session Interval Too Small to the client protocol entity, which contains its minimum acceptable refresh interval in the 422's Min-SE header [Donovan02]. The third INVITE contains all the necessary information and results in a 200 OK. Observe that the response to the second INVITE may have been a 403 Forbidden if the aggregator could not authenticate the user. This would have ended the interaction as well, but with a negative outcome.

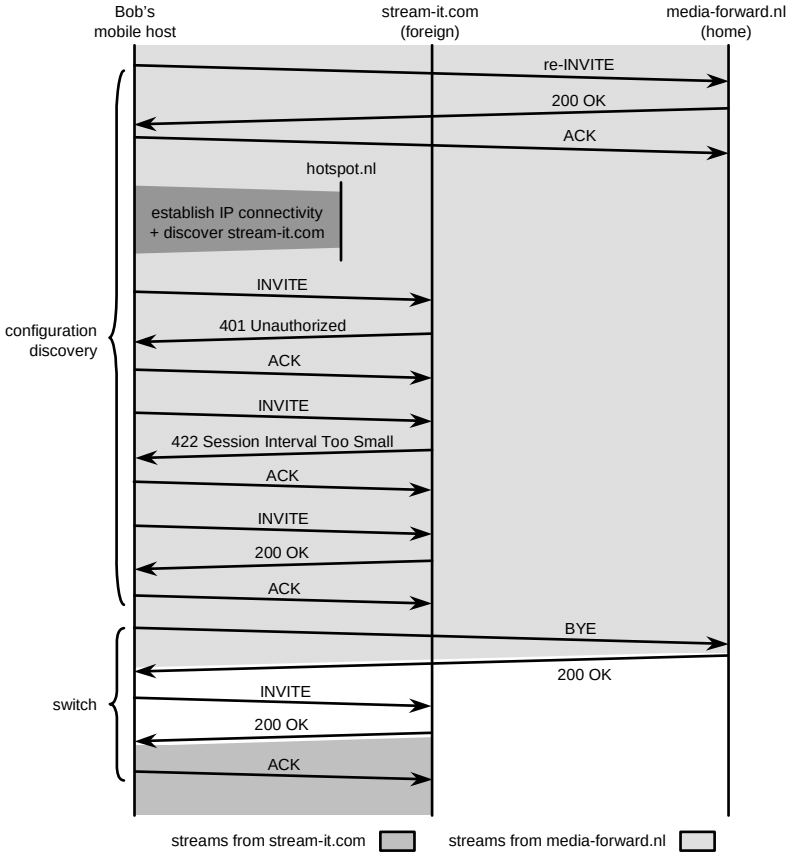


Figure 3-24. Multi-round configuration discovery interaction with an aggregator.

3.7.5 Switching

To switch between SIP servers, the client protocol entity transmits an INVITE to the target media server and a BYE to the current media server. *Figure 3-24* illustrates this at point B in *Figure 3-1* for a break-before-make switch. The INVITE and the BYE contain the Admission-Authorization-Token so that the media servers can verify that the user has been authenticated. The SDP payload contains the name of the channel and a description of the selected 'best' configuration of the channel. The SDP must not contain an a=inactive line.

A 200 OK response from the target media server indicates that the media server is transmitting the channel to the mobile host.

Change of IP address

Depending on the signaling protocol, the client protocol entity may be able to reuse an existing signaling association with a media server if it executes

an intra-media server switch (i.e., the target media server is the same as the current media server). SIP for instance enables mobile hosts to change the address of a mobile host without tearing down the signaling association (e.g., using the Contact header in a SIP re-INVITE messages [Wedlund99]). Of course, the media forwarder on the media server that transmits the actual multimedia packets (see Section 3.4.3) must also be able to deal with such address changes.

3.7.6 Testbed

We realized the SIP implementation of the ALIVE protocol using the Open SIP stack (version 1) [OpenSIP]. The Open SIP stack is written in C and also contains an SDP parser.

Figure 3-25 shows the small-scale testbed in which we deployed the implementation. Our objective was to realize inter-aggregator switches using multiple networks. The main components of the testbed are a laptop, an aggregator server, a Radius server (not shown in *Figure 3-25*), a fixed Ethernet, an 802.11 network, and a UMTS network. The laptop represents a mobile host in the ALIVE system and is equipped with an 802.11 interface (an Orinoco 802.11b Gold card), a fixed Ethernet interface, and a UMTS interface (via Bluetooth over USB). The aggregator server 'hosts' three aggregator domains and connects to the fixed part of the network. The Radius server represents a user's home aggregator and is based on the software of the Free Radius project [FreeRadius]. The Ethernet, the 802.11 LAN, and the UMTS network are three different subnets and represent three different access providers. All the machines the testbed use Linux.

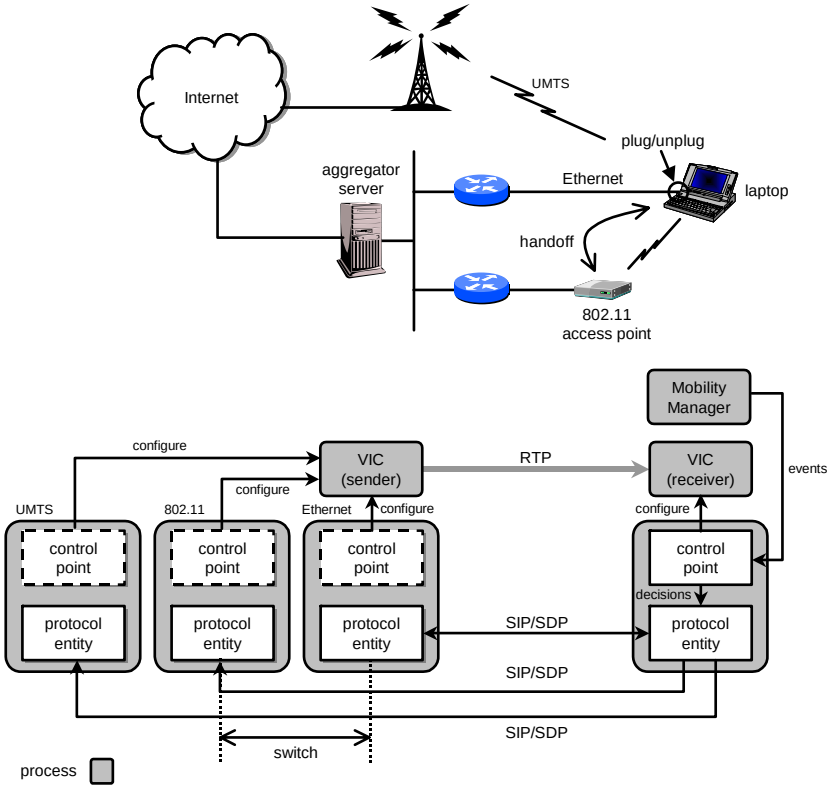


Figure 3-25. Testbed.

Aggregator Server

The aggregator server 'hosts' three aggregator domains in the form of three processes that each represent a front-end. Each process is bound to one network (Ethernet, UMTS, or 802.11) and consists of a control point and the ALIVE protocol (see Section 3.7) on top of a SIP user agent server. We bound the front-ends to a network by appropriately configuring the laptop (i.e., we did not use a protocol like DHCP to discover the front-ends). The front-end control points are virtually empty in our implementation and are an item of future work.

The front-end processes authenticate users with the Radius server and communicate with that server via the Ethernet. We configured the Radius server such that it returns a description of a user's allowed configurations if it can authenticate a user. We did however not implement the conversion to the configurations supported by the three foreign front-ends (i.e., the virtual environment of home configurations), which would be a task of the front-ends' control points.

The aggregator server also contains a process that runs the well-known video conferencing tool VIC [VIC] as a multimedia server. The three front-

ends use the VIC process to deliver a channel, which means that the three aggregators essentially use a shared pool of media servers. Of course, this is a situation that will not occur in practice.

To deal with IP addresses changes, we extended VIC such that the front-end processes can dynamically change the IP address to which the VIC server sends it packets. The extension also enables the front-ends to dynamically change the actual configuration in which VIC transmits a channel, specifically by changing the bitrate of the stream, its framerate, and its 'quality' (the latter is a VIC-specific metric).

Laptop

The laptop runs three processes, one of which executes the client control point and the client-side of the ALIVE-over-SIP protocol. The other two processes execute VIC (as a client) and a mobility manager [Peddemors04] that keeps track of the state of the laptop's network interfaces (e.g., which interfaces currently have link-layer connectivity). We attached the Open SIP software to a library that enables the stack to transmit a specific SIP message via a specific interface. This also required us to create three routing tables on the laptop, one for each of the laptop's interfaces. We furthermore modified VIC such that it can receive a multimedia stream via a specific interface.

The mobility manager sends events to the client protocol software (e.g., network disconnects), which then result in the ALIVE protocol being executed. For example, unplugging the Ethernet cable (see *Figure 3-25*) will result in the mobility manager informing the client control point that the Ethernet connection has gone down. As a result, the control point on the mobile host will invoke the ALIVE-over-SIP protocol to discover the available configurations of the aggregators bound to the 802.11 network and the UMTS network. After collecting their responses, the client control point for instance decides to switch to the aggregator bound to the wireless LAN, which means that it sends an establishment message (an INVITE) to it via the 802.11 network. This message configures the VIC server with the address of the laptop's wireless LAN interface instead of with the laptop's Ethernet interface and also changes the configuration in which the server is transmitting a channel.

3.8 Related Work

Several papers in the literature consider hosts that switch from one server machine to another (e.g., [Dutta02, Trossen03, Roy02, Xu00, Kim01]), but most of them do not investigate switches between the servers of different access-controlled domains (i.e., inter-aggregator switches). The

exception is [Trossen03], but their business network is application-neutral and is not based on agreements. In addition, their system puts most of the switching responsibilities in access routers instead of on mobile hosts.

The ACT Framework

Trossen and Chaskar [Trossen03] consider mobile hosts that need to dynamically connect to or switch between application-specific services (e.g., a transcoder) after an IP-level handoff (e.g., controlled by Mobile IP or SIP). Their value chain (discussed in the form of three scenarios) is comparable to ours. It consists of content sources, Supplementary Service Providers (SSPs), and wireless network operators. SSPs are comparable to our aggregators, except that SSPs are application-neutral, while our aggregators are specific to the distribution of real-time multimedia content. Similar to the aggregators in our model, SSPs can also be bound to specific network operators, in which case mobile hosts must be able to switch to another SSP when they leave a network operator's coverage area. A difference with our model is that content sources can also be bound to network operators, which is a case that we do not consider. Unlike our work, they do not consider agreements.

Trossen and Chaskar propose a framework that enables mobile hosts to switch between (or connect to) application-specific services such as those provided by their SSPs. Their Application Context Transfer (ACT) framework revolves around the notion of an application context, which is a block of application-specific information (e.g., an SDP description of a multimedia session) that is stored in the access router to which the mobile host attaches. In the event of a handoff, this router transfers the context to the host's new access router, which uses the information to decide how to continue the session. Trossen and Chaskar discuss several message sequence diagrams that suggest how the new access router can accomplish this. For example, if the mobile host was receiving a multimedia stream through a proxy server, then the new access router can decide to set up a bi-directional tunnel with the proxy so that the stream from the source will continue to flow through the proxy. Another possibility is that the new access router is responsible for discovering another source that provides the same content, for instance when the mobile host's current source is unavailable through the new access router (e.g., because the new access router belongs to another network operator).

While the ALIVE and ACT value chains are similar, the underlying systems are completely different. The main difference is that the ACT framework realizes its functions (e.g., discovery of SSPs) in access routers, while the ALIVE system puts these functions on mobile hosts. As a result, the ACT framework requires a more advanced and more complex router infrastructure (e.g., to maintain application-level state and to interact with

other access routers or sources). The ALIVE system requires no additional features from routers, which keeps the network simple. An advantage of the ACT approach is that it integrates with Mobile IP into a single solution. Finally, the ACT framework may also involve per-user interactions between the new access router and the content source, which might negatively affect the scalability of the system when delivering real-time multimedia content to a large number of mobile users (server implosion).

MarconiNet

The MarconiNet system of Dutta et al. [Dutta02] consists of affiliate domains that receive streams from radio or TV broadcasters and forward them to mobile hosts using a set of media servers. Each media server is responsible for one subnet and delivers streams to mobile hosts through locally scoped IP multicast groups. Mobile hosts switch from one server to another by switching to another multicast group. In [Dutta02], the authors discuss a signaling protocol that realizes such handoffs in combination with DiffServ-based QoS control. Like our system, MarconiNet runs in a managed environment with agreements between affiliates and radio or TV broadcasters. Their affiliates are similar to our aggregators. One of the main differences with our work is that Marconinet only considers handoffs between servers of the same affiliate and that they do not explicitly distinguish application and network-level roles. As a result, they miss most of the agreements of our model (e.g., application-level roaming agreements). Other differences are that we explicitly consider multi-homed mobile hosts and that their approach does not include the notion of a configuration. Our system therefore differs considerably from theirs. We do however not cover security issues, which Dutta et al. do.

Switching Between Transcoders

Roy et al. [Roy02] discuss a system that enables mobile hosts to seamlessly switch between two transcoding servers. They accomplish this by migrating the state of a transcoding session (e.g., information to reconstruct the next frame from the source at the target transcoder) from one server to another. The authors discuss three types of inter-server protocols that can be used for this purpose. We consider this work complementary to ours.

Internet Media Guides

Our work can also be considered from an Internet Media Guide (IMG) perspective [Nomura03]. The MMUSIC group of the IETF is currently looking into a framework for the distribution of IMGs to a potentially large number of (mobile) users. They define an IMG as a structured set of descriptions of multimedia sessions (e.g., in SDP) and distinguish IMG senders, IMG transceivers, and IMG receivers. An IMG transceiver receives

IMGs from senders, optionally modifies the IMGs, and forward them to IMG receivers. In our work, a multimedia session is a multimedia channel being transmitted at a certain configuration. Sources are IMG senders, aggregators are IMG transceivers, and mobile hosts are IMG receivers. Aggregators can be considered IMG transceivers because they bundle channels from sources and because they can offer channels to mobile users at other configurations than the sources from which they receive the channels (see Section 2.2). The work we presented in this thesis addresses at least two of the requirements in [Nomura03]. First, [Nomura03] requires that IMG receivers are allowed to communicate with multiple IMG senders simultaneously. Our (multi-homed) mobile hosts communicate with multiple IMG senders because aggregators are IMG transceivers and IMG transceivers are also IMG senders. A second requirement is that it must be possible to deliver customized IMGs to receivers. The front-ends of our aggregators do exactly this because they determine sets of available configurations on a per-user basis (albeit limited by the configurations that an aggregator supports).

ServiPoly

Like in our work, Xu et al. [Xu00] also consider the delivery of multimedia services in multiple configurations (they call this feature service polymorphism). In their ServiPoly system, clients request a multimedia service from a server. Clients include their available resources in the request (e.g., battery power and available bandwidth), which servers use to select a service configuration. Servers then deliver such a service configuration directly or through an intermediary proxy server. Proxy servers may be part of different domains and are therefore comparable to our media servers. While it is not the main focus of their paper, Xu et al. suggest that mobile hosts can handoff to a proxy server of another intermediary domain by resubmitting their request for a multimedia service. The original server (or a replica thereof) would then select a new configuration and a new proxy server for the client. This is similar to our system, except that their servers possess most of the intelligence that we put on mobile hosts. ServiPloy furthermore delivers configurations tailored to individual clients, which is probably less scalable for live multimedia channels with a large number of receivers.

Distributed HTTP Proxies

Kim, Lee, and Chung [Kim01] discuss a system of distributed proxy servers that deliver web pages from a web server to mobile hosts. The proxies are responsible for transcoding (e.g., reducing the size of resolution of images) and caching and each serve a specific set of base stations. The latter means that mobile hosts need to switch to another proxy server when they leave

their current proxy's service area, which is something that may happen in the middle of an HTTP session (i.e., when the mobile host is receiving a web page).

Mobile hosts are responsible for switching to another proxy. They use a table to map the ID of a base station to a proxy server and initiate the switch by sending a handoff message to the target proxy server. The handoff message contains the HTTP request that the mobile host used to start the HTTP session with the old proxy server, the URLs of the files that the host was able to receive from the old proxy, and the number of bytes it received of each file. This information enables the target proxy server to determine if the mobile host has received all the files associated with the web page it requested from the old proxy.

The target proxy can retrieve transcoded images that the mobile host has not yet received from the old proxy. Proxy servers exchange such images through synchronization request and response messages. Alternatively, a proxy server may go directly to the web server to retrieve missing files.

The main difference between the work of Kim et al. and our work is that their work lacks a well-defined business network (cf. Chapter 2). Another major difference is that they focus on HTTP communications rather than on real-time streaming, which yields completely different system requirements. Handoff delay, for example, is less of an issue in their system than in our system (they reported average handoff delays in the range from 5 to 20 seconds). Another difference is that our system builds on standard IETF protocols, while their handoff and synchronization messages do not seem to follow a standard. Kim et al. do however cover state transfer between proxy servers (cf. [Roy02, Trossen03]), which is something that we have not considered.

The main similarities are that they also use a system of distributed proxy servers to deliver information to mobile hosts. In addition, their design also seems to put most of the system's intelligence on mobile hosts, although they do not specifically articulate that.

Switching on the Fixed Internet

Karrer and Gross [Karrer01] discuss an application-level system that enables *fixed* Internet clients to switch between (proxy) servers that produce the same video stream. They consider such switches an alternative to a server adapting a video stream itself (e.g., by dropping frames). In their system, clients are responsible for switching to another server and also for detecting events that may lead to such a switch.

The primary goal of the work of Karrer and Gross is to enable clients to transparently switch between servers. To accomplish this, they investigate how they can offer a constant data stream to a client's player during a

switch. They assume that a stream consists of a sequence of numbered packets and that servers use the same numbering for the same stream.

Karrer and Gross' solution uses clients that first store the packets they receive in a queue and then feed them to a video player. Their central notion is that of a start packet, which is the first packet that a client wants to receive from a target server. A client informs a target server of the number of its desired start packet when it initiates a switch to that server.

Start packets can be used to repair packet loss incurred by the stream coming from the old server. A client could for instance ask the target server to begin with a packet that the client is about to play back (i.e., more at the head of the queue), which would give the client the opportunity to receive any lost packets from the target server. Conversely, if a client hardly looses any packets from the old server, then it may request the target server to begin with a packet that is further away from being played back (i.e., more toward the queue's tail). This would reduce the number of duplicate packets, which reduces bandwidth consumption but offers less opportunity to restore missing packets. Observe that clients may decide to temporarily receive a same stream from the old server as well as from the new server to improve the quality of the switch.

The continuity of the stream arriving at a client is furthermore influenced by the delay between the client and the target server. The authors demonstrate this by having a client in Europe switching between a server in the US and a server in South America.

Karrer and Gross use start packets in combination with the delay between the client and the target server to develop four switching policies. One of these policies says that if packet loss on the path from the old server is 'high' and the delay to the target server is 'large', then the client should use a start packet that is currently at the head of the queue.

The work of Karrer and Gross differs form our work in that they only consider switches between servers instead of between access-controlled domains (our aggregators). In addition, they only consider the switches itself and not the signaling interactions that precede it (e.g., to discover which servers are available and which streams they offer).

Others

Hsieh et al. [Hsieh03] discuss a receiver-oriented TCP-clone that is able to hand a TCP connection off from one server to another. While they also consider multi-homed mobile hosts, their work is at the transport layer, which makes it quite different from ours.

[Snoeren01b, Sultan02] consider handoffs between servers for other reasons than mobility, for instance to increase the availability of a service (e.g., handoff to another server when the current server gets overloaded). They transfer TCP state (e.g., the sequence number of the last successfully

acknowledged data segment) and some application-level state to resume a TCP connection (e.g., for HTTP applications) at the target server at exactly the same place where it left off at the original server. A similarity with our work is that the clients in [Snoeren01b] are responsible for selecting a target server just like our mobile hosts are responsible for selecting a target configuration, aggregator, and media server.

Chapter 4

Analysis

In this chapter, we analyze overhead of the ALIVE protocol, specifically in terms of the extra delay it introduces. We concentrate on environments with 802.11 hotspots, where the ALIVE protocol typically comes into play at the edge of an 802.11 cell. At these edges, we experiment with the delays introduced by the ALIVE protocol, which may be substantial as a result of the exponential back-off retransmission scheme used by the Session Initiation Protocol (SIP) to recover from packet loss. As we have seen in Chapter 3, we implemented the ALIVE protocol as a thin protocol layer on top of SIP.

We first outline our approach (Section 4.1) and take a look at the different types of delays involved in a typical switch (Section 4.2). Next, we discuss our experiments, which concentrate on the retransmission behavior of SIP transactions under various 802.11 network conditions (Section 4.3). After that, we describe our measurement set-up (Section 4.4) and discuss the results of our experiments (Section 4.5). We conclude this chapter with a description of related work (Section 4.5.7).

4.1 Goal and Approach

The goal of our analysis is to determine the overhead introduced by the ALIVE protocol in a contemporary network environment consisting of 802.11 hotspots and overlaying UMTS or GPRS networks [Køien03, Banerjee04, Zhuang03]. We do not consider alternative hotspot technologies such as HIPERLAN [Doufexi03], GSM micro-cells [Tripathi98], and infrared [Brewer98].

We zoom in on mobile hosts that receive a channel via their 802.11 interface and execute a switch as a result of moving into the coverage area of another 802.11 access provider. The reason for concentrating on this type of scenario is that it might result in a mobile host being unable to receive the channel for an extended period of time. This has two causes:

- An 802.11 handoff (in this case to an access point of the target 802.11 access provider) temporarily disconnect the mobile host

from the 802.11 infrastructure [Velayos03, DeCleyn04, Vatn03, Mishra03]; and

– Before initiating the switch, the switching controller on the mobile host will first invoke the ALIVE protocol to discover the available configurations of local aggregators on the target network (configuration discovery). As a result, the mobile host will be temporarily disconnected from the aggregator infrastructure as well.

The detachment from the aggregator infrastructure may last for quite some time because the execution of the ALIVE protocol at the edge of the target 802.11 network may introduce significant delays. This is because 802.11 links are known to be lossy under certain circumstances [Eckhardt96, Hoene03, Aguayo04, Punnoose01] and because we implemented the ALIVE protocol on top of SIP. SIP uses an exponential back-off retransmission scheme to recover from packet loss [Rosenberg02a] and uses a default back-off time of 0.5 seconds. Observe that overlay networks such as UMTS or GPRS typically do not trigger SIP retransmissions as a result of packet loss because they use a semi-reliable link-level protocol (the Reliable Link Protocol, RLP [Banerjee04]).

As a result of the 802.11 handoff and the delay introduced by the ALIVE protocol, there is a risk of the mobile host's playout buffer (see Section 3.2) depletes before the switch completes. To avoid this situation, the delay introduced by the 802.11 handoff plus the delay of the ALIVE protocol must be smaller than or equal to the amount of audio and video in the playout buffer (in seconds). If this is the case, then we speak of a *smooth switch*, which means that the playout buffer can continue to feed packets to the player during and after the switch (cf. [Karrer01]).

Figure 4-1 shows an example of the type of scenario we focus on in this chapter. The example is an extension of the example in Figure 3-1 and shows Bob moving through two hotspots while receiving CNN TV. Bob's mobile host receives CNN TV from stream-it.com in both hotspots and executes a handoff from hotspot2.nl to hotspot1.nl at point C. During the handoff, Bob's mobile host will be temporarily disconnected from the 802.11 infrastructure. It is not until after the handoff that it will be able to interact with stream-it.com, the local aggregator of hotspot1.nl.

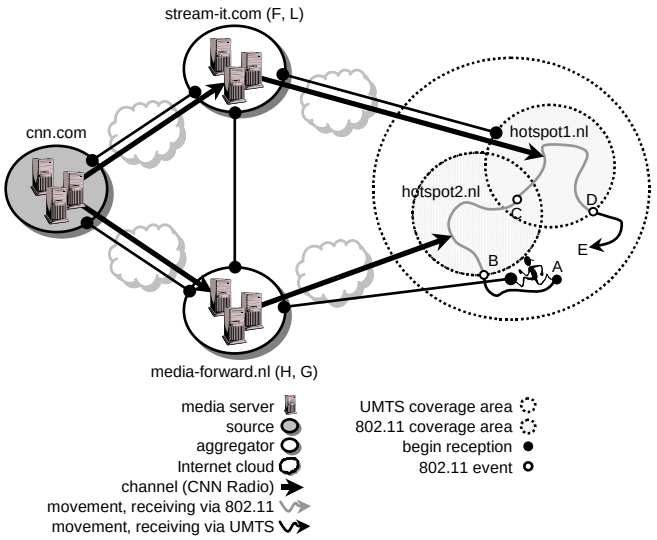


Figure 4-1. Roaming through two hotspots.

We assume that mobile hosts always have IP connectivity on the UMTS overlay network ('always-on'). This means that a switch to an aggregator that is available through that network can usually take place quickly (e.g., at point D in *Figure 4-1* when Bob roams out of hotspot1.nl's coverage area) and is typically not time-critical. A switch from an aggregator on the overlay network to an aggregator on the 802.11 network (e.g., at point B) is not time-critical either, in this case because the mobile host does not need to disconnect from the infrastructure. Switches between aggregators that belong to overlay networks of different access providers (e.g., between UMTS networks in bordering countries) are outside the scope of this thesis.

4.2 Delay Components

In this section, we provide an overview of the various delay components that play a role in the ALIVE system when a mobile host moves into the coverage area of another 802.11 access provider (see Section 4.1). The goal of this overview is to put the delay introduced by the ALIVE protocol into perspective, specifically to be able to contrast it with the other delay components in the system (e.g., the delay to execute an 802.11 handoff).

For reasons outlined in Section 4.1, we concentrate on the operation of the ALIVE protocol on 802.11 links. We isolate the delays caused by 802.11 links and use the data we found in the literature to provide best and worst-case estimates for the other delay components. We assume that backbone links are reliable (cf. the packet loss statistics on the Abilene network [Abilene04]) and ignore local processing delays. The only

exception is the delay to authenticate a user, which may be substantial as observed in [Mishra04, Pack02].

We distinguish two overall delay components: the *ALIVE protocol delay* (for configuration discovery and switching) and the *IP handoff delay*. An IP handoff connects a mobile host to another 802.11 network and establishes IP connectivity on that network. An IP handoff includes the execution of the 802.11 handoff, the authentication of the user on the target network, and so on.

In the host architecture of Section 3.4, the connectivity handler is responsible for running the protocols to execute an IP handoff (e.g., DHCP [Droms99] to get an IP address for an interface), whereas the ALIVE protocol entities are responsible for executing the ALIVE protocol.

We first consider the IP handoff delay (Section 4.2.1) and then the ALIVE protocol delay (Section 4.2.2). We conclude this section with a discussion on the overhead of the ALIVE protocol (Section 4.2.3), which we express as the ratio of the ALIVE protocol delay and the IP handoff delay.

4.2.1 IP Handoff Delay

Figure 4-2 shows the message sequence diagram of an IP handoff. The IP handoff delay consists of five subdelays: a detection delay (not shown in *Figure 4-2*), an 802.11 handoff delay, a network-level authentication delay, and an interface configuration delay.

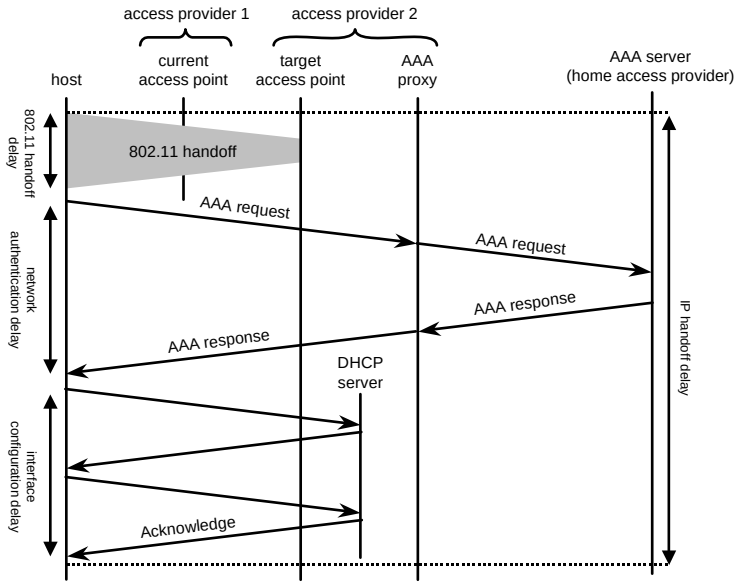


Figure 4-2. Delay components to execute an IP handoff to another 802.11 network.

Detection Delay

In general, the detection delay is the delay it takes the switching controller of a mobile host to detect that an event has occurred (e.g., on one of the host's interfaces). The detection delay depends on the mechanism used to detect events (e.g., on RTCP Receiver Reports [Schulzrinne96a] that are generated every 5 seconds to report packet loss rates and other information pertaining to the quality of a stream).

802.11 Handoff Delay

The 802.11 handoff delay is the delay to connect a mobile host to another 802.11 access point. Average values for 802.11b handoff delays that have been reported in the literature vary from around 40 to almost 600 milliseconds [Mishra03, Vatn03, Shin04, Velayos03]. Actual 802.11 handoff delays can vary considerably, for instance as a result of the 802.11 hardware that the mobile host and the access points use [Mishra03, Vatn03], and the amount of competing traffic on the target network, either as a result of traffic generated or received by other hosts on the target access point's frequency [Vatn03, Velayos03], or as a result of traffic from adjacent frequency channels [Shin04].

The major delay component in an 802.11 handoff is the time that a mobile host spends scanning for a target access point. The scanning delay constitutes about 90% of the entire 802.11 handoff delay [Shin04, Velayos03] and must take place after the mobile host disconnected from its current access point. One solution to speed up the handoff process is to selectively scan the 802.11 channels (11 channels in the US, 13 in most of Europe) for available access points, which reduces the handoff delay to an average of around 150 milliseconds [Shin04].

In the rest of this chapter, we assume that average 802.11 handoff delay lies between 40 and 600 milliseconds:

$$40 \leq t_{802\_ho} \leq 600$$

where t_{802_ho} is the 802.11 handoff delay.

Network-level Authentication Delay

The network-level authentication delay is the time it takes to authenticate a user on the target 802.11 network. The network-level authentication delay is highly deployment dependent. It for instance depends on the authentication mechanism (e.g., password or certificates) and on the protocol used in the target network to exchange authentication keys, encryption keys, and so on (e.g., 802.1x [Mishra04, Pack02]). In an inter-access provider handoff, the target access provider will typically authenticate

a user with his home access provider, which requires a AAA interaction with that provider [Køien03, Kwon02].

In this chapter, we assume that the mobile host authenticates a user with an 802.11 network through a single request-response interaction with a AAA proxy of the target 802.11 provider (see *Figure 4-2*) [Kwon02]. We furthermore assume that the interaction between the AAA proxy and the user's home access provider also consists of a single request-response pair, typically using a protocol like Diameter [Calhoun03].

As a result, the network-level authentication delay equals:

$$d(host, accessPoint) + d_{rtx}(host, aaaProtocol) +$$
$$2*d(aaaProxy, accessPoint) +$$
$$2*d(aaaProxy, homeAggregator) +$$
$$d_{auth}(homeAggregator) +$$
$$d(accessPoint, host) + d_{rtx}(aaaProxy, aaaProtocol)$$

where d(src, dst) is the *one-way* delay to convey a message from source src to destination dst, d_{auth}(e) the delay to authenticate a user at entity e, and d_{rtx}(src, protocol) the delay introduced by a source src to reliably deliver a message over a network path that involves a wireless link.

d_{rtx}(src, protocol) depends on the specific higher-layer protocol that the source uses to deliver the message, typically by means of retransmissions (cf. SIP). The higher-layer protocol could for instance use an exponential back-off retransmission scheme, in which case d_{rtx}(src, protocol) would equal b + 2*b + 4*b + … n*b, with b being the back-off time and n a power of two. Notice that d_{rtx}(src, protocol), d(host, accessPoint), and d(accessPoint, host) depend on the charateristics of the wireless link at the moment the source (re)transmits a message (e.g., in terms of SNR, transmission rate, and so on).

The delays on the up and down links between the host and the AAA proxy may be different as a result of the varying conditions of the 802.11 radio link, which is why we have represented them as two separate delay components (i.e., d(host, acessPoint) and d(accessPoint, host)). We have assumed that the delays on the fixed network are symmetric (see the one-way delay statistics of the Abilene network [Abilene04]), which is why the round-trip delay on the fixed network is twice the one-way delay (e.g., 2*d(aaaProxy, accessPoint)).

We will reuse the above notations and assumptions throughout this section.

We estimate the authentication delay at a home access provider (d_{auth}(homeAggregator)) to lie between 1.1 second and 50 milliseconds, which are values reported in [Mishra04] for intra-domain domain

authentication. The 50 milliseconds is the average of an optimized 802.1x authentication mechanism.

For the delay between the AAA proxy and the home access provider (d(aaaProxy, homeAggregator)) we assume an average of at most 38 milliseconds, which is the average one-way delay on the Abilene network from New York City to Los Angeles (measured from April 4 till May 4, 2004) [Abilene04]. The minimum delay between the proxy and the home access provider is 0 milliseconds, which occurs when the proxy belongs to the home access provider (i.e., the target access point belongs to the user's home access provider).

The access point and the AAA proxy are part of the same access provider, so we assume an average one-way delay of 2 milliseconds between these two entities [Kwon02].

Based on our own measurements (see Section 4.5), we assume a best-case one-way delay across the 802.11 link (i.e., for d(host, accessPoint) and d(accessPoint, host)) of 12 milliseconds. This is on an 802.11b network at a 1 Mbps transmission rate, which is the typical rate at the edge of an 802.11 cell. The 12 milliseconds was measured at the socket-level and therefore also includes operating system delays.

In the best case, the host and the AAA proxy also do not need to retransmit any messages, which means that d_{rtx}(host, aaaProtocol) and d_{rtx}(aaaProtocol, host) are zero.

Summing up, the network-level authentication delay is bounded by:

$$78 \leq t_{net\text{-}auth}(aaaProtocol, homeAccessProvider) \leq 1180 + d(host, accessPoint) + d_{rtx}(host, aaaProtocol) + d(accessPoint, host) + d_{rtx}(aaaProxy, aaaProtocol)$$

where $t_{net\text{-}auth}$(aaaProtocol, homeAccessProvider) is the total network-level authentication delay for a specific network access protocol and a specific home access provider.

Observe that in our analysis authentication takes place before the mobile host receives an IP address (as is the case in 802.1x [Gast02, Mishra04, Pack02]). Alternatively, authentication can also take place afterwards [Kwon02].

Interface Configuration Delay

The interface configuration delay is the time it takes to configure a mobile host's 802.11 interface on the target network. At a minimum, this means that the interface needs to be configured with an IP address and that it needs the IP and MAC addresses of a default router on the 802.11 network. In this thesis, we assume that DHCP [Droms99] gets the interface's IP

address and the IP address of the default router, while ARP is responsible for getting the MAC address of the default router [Kwon02].

On a fixed Ethernet, the average DHCP delay can vary from a few seconds to almost 15 seconds [Vatn98]. This is primarily the result of the way in which different DHCP stacks implement Duplicate Address Detection (DAD) [Vatn98], which is a procedure in which a host verifies that no other hosts on the network are using a particular IP address. For stateless autoconfiguration in IPv6 (i.e., without DHCP), the average DAD delay is 1.5 seconds [Nakajima03].

In this chapter, we assume that DHCP skips DAD (as suggested by [Vatn98] and [Nakajima03]), which reduces the DHCP delay to the link delay. Since a DHCP sequence consists of four messages (a Discover, an Offer, a Request, and an Acknowledge, see *Figure 4-2*), the DHCP delay is

$$2*d(host, accessPoint) + 2*d_{rtx}(host, dhcp) + 2*d(accessPoint, host) + 2*d_{rtx}(dhcpServer, dhcp) + 4*d(accessPoint, dhcpServer)$$

We also assume that the DHCP server is co-located with the default router of the 802.11 network. As a result, the mobile host can learn the router's MAC address from the messages sent by the DHCP server, which avoids the extra ARP round-trip over the link [Vatn98]. In this case, we can furthermore assume that the average one-way delay between the access point and the DHCP server is 2 millisecond since both will be on the same network. Again assuming a best-case one-way over-the-air delay of 12 milliseconds, the interface configuration delay is bounded by:

$$56 \leq t_{if\text{-}config}(dhcp) \leq 2*d(host, accessPoint) + 2*d_{rtx}(host, dhcp) + 2*d(accessPoint, host) + 2*d_{rtx}(dhcpServer, dhcp) + 8$$

where $t_{if\text{-}config}(dhcp)$ is the interface configuration delay using DHCP.

The configuration delay is usually followed by a *location update delay* (cf. Mobile IP [Solomon98], SIP [Wedlund99], and so on), which is the time it takes to inform correspondent hosts and 'home agents' of the mobile host's new IP address. In the ALIVE system, these location updates are only necessary if the switching controller on the mobile host has used the configuration notification service to subscribe to the events of at least one aggregator (see sections 3.4.6 and 3.6.2). We assume that location updates in the ALIVE system take place after the completion of a switch, which means the associated delay is not part of our analysis. Observe that a location update might be more critical for other applications that run on the mobile host (e.g., a telephony application).

Aggregator Discovery Delay

The aggregator discovery delay is the time it takes to discover one or more local aggregators on the target 802.11 network. We assume that the connectivity handler piggybacks the aggregator request and response messages onto DHCP messages. In particular, we assume that DHCP Acknowledge messages will contain the URIs that point a mobile host to the local aggregators on the network [Schulzrinne02]. As a result, the aggregator discovery delay does not any additional delay in our analysis.

4.2.2 ALIVE Protocol Delay

Figure 4-3 shows the message sequence diagram of the ALIVE protocol in case it detects a new local aggregator on the target 802.11 network. To keep the figure readable, it does not show the SIP ACK messages. *Figure 4-3* also assumes that the mobile host's current aggregator is no longer reachable on the target 802.11 network, which means that the switching controller on the mobile host cannot send a release message to the current media server to release the multimedia session (see Section 3.4). To simplify our analysis, we assume that the switching controller decides not to discover the capabilities of the local aggregators on the target 802.11 network. As a result, we omit capability discovery from our analysis (*Figure 4-3* therefore shows the capability discovery interactions as dashed arrows).

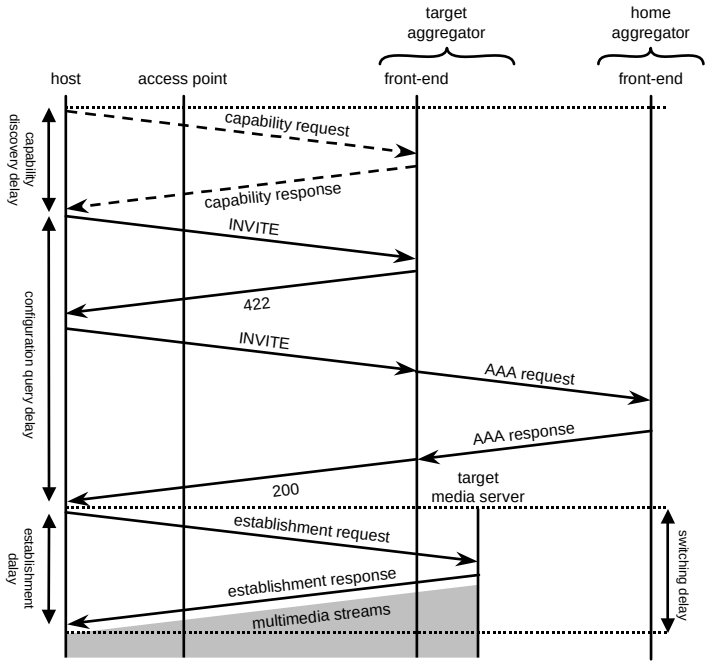


Figure 4-3. Delay components of the ALIVE protocol on an 802.11 network.

Configuration Query Delay

The configuration query delay for a specific aggregator depends on whether the switching controller has already authenticated the user with the aggregator. If this is not the case, then the switching controller first sends an INVITE (a combined authentication-configuration request, see Section 3.7) to the aggregator's front-end (see *Figure 4-3*). The ALIVE protocol entity on the mobile host may need to resubmit the INVITE request at most twice before it will be able to receive a 200 OK. Specifically, it will need to resubmit the INVITE if the front-end does not accept the refresh interval in the INVITE, or when it did not include the user's credentials in the INVITE. We will assume that the latter will not occur, which means that the maximum configuration discovery delay involves two round-trips with the front-end and one-round trip with the home aggregator:

$$2*d(host, accessPoint) + 2*d_{rtx}(host, sip) +$$
$$2*d(accessPoint, host) + 2*d_{rtx}(frontEnd, sip) +$$
$$4*d(frontEnd, accessPoint) +$$
$$2*d(frontEnd, homeAggregator) +$$
$$d_{auth}(homeAggregator)$$

If a switching controller has already authenticated a user with an aggregator (not shown in *Figure 4-3*), the configuration query delay is the delay between the transmission of a re-INVITE (configuration request) and the arrival of the 200 OK (configuration response). This situation can for instance occur when the same local aggregator was also bound to the old 802.11 network. In this case, the configuration query delay equals:

$$d(host, accessPoint) + d_{rtx}(host, sip) +$$
$$d(accessPoint, host) + d_{rtx}(frontEnd, sip) +$$
$$2*d(frontEnd, accessPoint)$$

We assume that the local aggregators on the target 802.11 networks are in the 'network vicinity' of the access provider (i.e., a limited hops a way from the access provider's network). We therefore assume a maximum average delay between the access point and the front-end of the local aggregator of 10 milliseconds, which is the average rounded-off one-way delay on the Abilene network from New York City to Chicago [Abilene04]. In the most extreme case, the aggregator and the access provider will be co-located in one domain (cf. the Supplementary Service Providers in [Trossen03]), in which case we assume that the average one-way delay between the access point and the aggregator's front-end is 2 milliseconds.

As before (see Section 4.2.1), we assume a best-case one-way delay across the wireless link of 12 milliseconds, a minimum for d_{rtx} of 0 seconds,

a maximum one-way delay to the user's home aggregator of 38 milliseconds, and a maximum authentication delay of 1.1 seconds. Using the authenticated case as the best case, the configuration query delay on an 802.11 network is bounded by:

$$28 \leq t_{query} \leq 2*d(host, accessPoint) + 2*d_{rtx}(host, sip) + 2*d(accessPoint, host) + 2*d_{rtx}(frontEnd, sip) + 1216$$

where t_{query} is the configuration query delay.

The ALIVE protocol entity on the mobile host typically transmits multiple INVITEs to multiple front-ends (see Section 3.6), in which case the total configuration query delay is determined by the highest query delay of the individual aggregators (e.g., as a result of varying conditions of the 802.11 link when the 200 OKs arrive at the access point). The ALIVE protocol entity on the mobile host can however cap the discovery time at a certain threshold (see Section 3.4).

Switching Delay

In general, a switching controller can execute a switch in a number of ways, for instance in a break-before-make fashion (first release the multimedia session with the current server, then establish a new session with the target media server) or in a break-before-make fashion (the other way around). In our analysis, we assume that the switching controller can no longer reach the current media server via the target 802.11 network and therefore only transmits an establishment request to the target media server. As a result, the switching delay is the delay between the transmission of the establihsment request and the arrival of the response, which corresponds to twice the one-way delay from the host to the target media server. Also assuming that the target media server is a SIP server, the switching delay equals:

$$d(host, accessPoint) + d_{rtx}(host, sip) + d(accessPoint, host) + d_{rtx}(targetMediaServer, sip) + 2*d(targetMediaServer, accessPoint)$$

We assume that the media servers of an aggregator are co-located with the aggregator's front-end, which means that the maximum one-way delay between a media server and an access point is 10 milliseconds. The minimum one-way delay is 2 milliseconds (if the aggregator and the access provider are co-located in one domain). The switching delay is therefore bounded by

$$28 \leq t_{switch} \leq d(host, accessPoint) + d_{rtx}(host, sip) +$$

$$d(accessPoint, host) + d_{rtx}(targetMediaServer, sip) + 20$$

where t_{switch} is the switching delay.

4.2.3 ALIVE Protocol Overhead

One way to calculate the overhead of the ALIVE protocol is to determine the ratio of the ALIVE delay and the best-case IP handoff delay. That is:

$$(t_{query} + t_{switch})/174$$

where 174 milliseconds in the sum of the lower bounds of t_{802_ho}, $t_{net\text{-}auth}$(aaaProtocol, homeAccessProvider), and $t_{if\text{-}config}$(dhcp).

If we use the minimum values for t_{query} and t_{switch}, then the ratio equals (28+28)/174, which is approximately 32%. That is, the best-case ALIVE delay is around 32% of the best-case IP handoff delay.

In the rest of this chapter, we analyze the non-best-case values of 2*d(host, accessPoint) (and vice versa) and d_{rtx}(host, sip), in particular under different 802.11 network conditions.

4.3 Experiments

As we have seen in Section 4.2.3, the configuration discovery delay of the ALIVE protocol depends on the delay on an 802.11 link and on the time it takes a SIP sender (a switching controller or a front-end) to deliver a message to a SIP receiver (i.e., d_{rtx}). Both of these factors in turn depend on the packet loss characteristics on the wireless link.

In the ALIVE system, SIP runs on top of UDP (see Section 3.7), which means that the messages of a transaction may get lost. To reliably execute transactions over UDP, SIP uses an exponential back-off retransmission mechanism [Rosenberg02a]. The default back-off time of this mechanism is 0.5 seconds, which means that the loss of a single SIP message results in a delay of 0.5 seconds, two consecutive losses in a delay of 0.5 + 2*0.5 seconds, three consecutive losses in 0.5 + 2*0.5+ 4*0.5 seconds, and so on. Delays of this magnitude would dominate the configuration discovery delay, in particular when aggregators cache authentication state to reduce the authentication delay and when mechanisms like selective scanning, pre-authentication, and selective DAD (see Section 4.2.1) are used at the network-level.

In the rest of this chapter, we therefore experiment with the retransmission behavior of SIP transactions, specifically over 802.11b links. The *goal* of our experiments is to determine how different network-level

packet loss parameters (e.g., signal to noise ratio and transmission rate) influence the retransmission behavior of a SIP transaction. We concentrate on mobile hosts that are at the edges of 802.11 cells, which is where the ALIVE protocol typically comes into play (see Section 4.1).

The results of our experiments enable mobile hosts to dimension their playout buffer such that switches can take place in a smooth manner, in particular during an 802.11 handoff. A spin-off is that our experiments might assist access providers in dimensioning their 802.11 infrastructure in such a way that it increases the probability that a mobile host can smoothly switch to another aggregator.

In this section, we take a more detailed look at SIP's retransmission scheme (Section 4.3.1) and discuss the 802.11 parameters that we vary in our experiments (Section 4.3.2).

4.3.1 SIP Retransmissions

SIP only retransmits messages when it runs on top of UDP. The retransmission scheme is exponential back-off with a default back-off timer of 0.5 seconds [Rosenberg02a]. A retransmitted SIP messages is a *duplicate* of the original message, which means that a retransmitted message carries the same dialog and transaction identifiers (see Section 3.7.2) as the original. *Figure 4-4* shows an example of SIP's retransmission behavior.

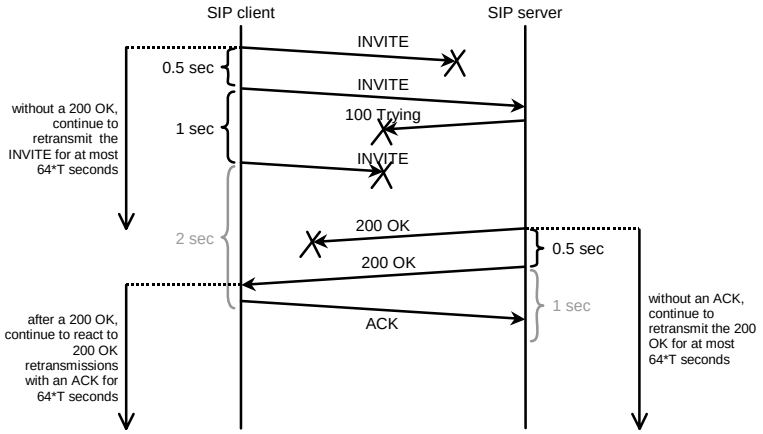


Figure 4-4. Example of SIP's retransmission behavior.

After it sent the original INVITE, a SIP client retransmits an INVITE for at most 32 seconds or until it receives a response (100 Trying or 200 OK), whichever happens first. As a result, a SIP client can retransmit an INVITE at most 6 times (i.e., transmit it 7 times), specifically at 0.5, 1.5, 3.5, 7.5, 15.5, and 31.5 seconds after the transmission of the original INVITE.

A SIP server retransmits an 200 OK for at most 32 seconds or until it receives an ACK, whichever happens first. A SIP server follows the same

retransmission scheme as a SIP client, except that it caps the back-off time at 4 seconds. As a result, a SIP server can retransmit a 200 OK at most 10 times (i.e., 11 transmissions) after the transmission of the original 200 OK (at 0.5, 1.5, 3.5, 7.5, 11.5, 15.5, 19.5, 23.5, 27.5, and 31.5 seconds after the transmission of the first 200 OK).

We consider a SIP transaction *failed* if (1) the client did not receive a 200 OK 32 seconds after it sent the original INVITE, or (2) when the server did not receive an ACK 32 seconds after it sent the original 200 OK. We note that the SIP RFC does not explicitly call the first case a transaction failure. In addition, if a 200 OK reaches the client but the ACK never gets back to the server, then the SIP RFC does not consider that a transaction failure either because an ACK is not part of an invite transaction that involves a 200 OK.

A SIP server transmits a provisional response when it receives an INVITE, either the first one or a retransmission. It does however not actively retransmit the provisional responses. Similarly, SIP clients only transmit an ACK when they receive a 200 OK (original or retransmission) and do not actively retransmit ACKs either.

4.3.2 802.11 Packet Loss

In general, packet loss on an 802.11 network depends on a wide variety of parameters [Eckhardt96, Hoene03, Aguayo04]. In this thesis, we experiment with four them on 802.11b links:

- *Signal strength*. The signal strength influences the probability that a message is lost or corrupted when it is in transit to its destination. The lower the signal strength, the higher the probability that a packet will get lost or corrupted [Hoene03, Punnoose01, Doufexi03, Aguayo04];
- *Retry limit*. 802.11 senders retransmit a frame until they get an acknowledgement from the receiver [Gast02]. The retry limit indicates how often a sender will retransmit unacknowledged frames before it considers them lost. The higher the retry limit, the lower the probability that the application will have to retransmit messages itself. However, higher values of the retry limit also increase the time it takes to get a message to the other end [Hoene03]. Higher retry limits might furthermore increase the medium access delay as 802.11 senders do not accept a new frame from the application until the previous one was either acknowledged or lost [Punnoose01].
- *Background traffic*. Since 802.11 is a shared-medium network, traffic from other hosts on the network can increase the medium access delay as well as the number of collisions [Aguayo04]. Both factors can increase the packet loss rate on the network.

– *Bit rate*. 802.11b supports four bit rates: 1, 2, 5.5, and 11 Mbps. The network is more reliable at 1Mbps than at 11 Mbps [Hoene03] as a result of the different modulation schemes that an 802.11 radio uses at different rates.

In our experiments, we use low signal strengths and usually set the transmission rate to 1 Mbps. This corresponds to the situation where a mobile host is at the edge of an 802.11b network. We use different retry limits and run experiments with and without background traffic. We refer to Section 4.5 for a discussion on the results.

Other sources of packet loss in 802.11 networks include spatial distribution, interference from other sources (e.g., Bluetooth [Punnoose01]), effects of multi-path, and so on. We refer to [Eckhardt96, Hoene03, Aguayo04] for an elaborate overview.

4.4 Measurement Set-up

This section discusses the set-up that we used to conduct our experiments. We first provide an overview of the set-up (Section 4.4.1) and then consider its basic operation (Section 4.4.2) and its physical arrangement (Section 4.4.3). We conclude with an overview of the hard and software we used (Section 4.4.4).

4.4.1 Overview

Figure 4-5 shows the high-level organization of our measurement set-up. The main components of the set-up are a SIP client and a SIP server that execute a series of SIP transactions over an 802.11b link. We refer to such a series of transactions as a *(transaction) run*. The other components in the set-up are a network sniffer and a traffic generator.

The whole set up is controlled by a script, which reconfigures the 802.11 network, the traffic generator, and the sniffer after each transaction run, and then starts a new run. The control script transmits its instructions via a fixed network (Ethernet) and connects to the other components through telnet connections.

The SIP client and SIP server are based on the Open SIP stack (version 1) [OpenSIP], which we enhanced with our own software to make measurements. The SIP client is a laptop with a Prism2 802.11b card. The SIP server is a PC with the same card, but configured to acts as an access point. The sniffer and the traffic generator are separate laptops, both with an Orinoco gold 802.11b card. The sniffer uses tcpdump [tcpdump] to capture packets; the traffic generator uses the tool jtg [jtg] to generate

traffic. The PC and the laptops all run on Linux. We refer to *Table 4-1* at the end of this section for more details.

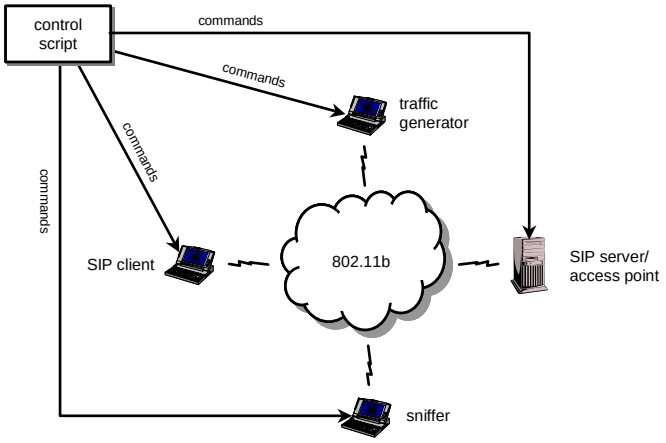


Figure 4-5. High-level organization of the measurement set-up.

4.4.2 Basic Operation

Figure 4-6 shows the basic operation of the control script, which runs on the SIP client.

The control script is a nested for-loop that reconfigures the 802.11b network at each iteration and then starts a new transaction run. To reconfigure the network, the script changes the signal to noise ratio (SNR) at the SIP client, the retry limit and the transmission rate at the SIP client and the SIP server, and the amount of background traffic injected into the network by the traffic generator (see Section 4.4.1).

To change the SNR at the client, the control script modifies the transmission power of the access point (the SIP server). Due to hardware/firmware limitations, we could not change the transmission power of the SIP client, which means that it is always transmitting at the default transmission power (-3 dBm). As a result, our set-up is *asymmetric*. This means that the INVITEs and ACKs of a transaction will generally arrive at the SIP server, but that the 200 OKs and 100 Tryings might be lost as a result of a low SNR value.

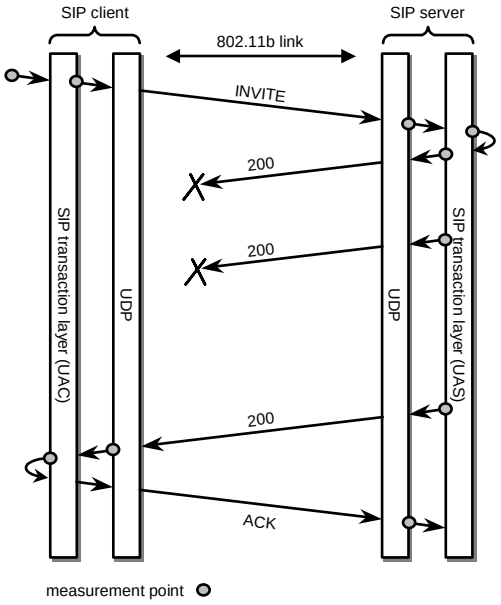
After it has reconfigured the network, the control script starts a new transaction run by starting the SIP server and the SIP client. At that point, the script also restarts the sniffer and the traffic generator. *Figure 4-6* shows that the control script transmits a start commands to the remote devices (SIP server, traffic generator, and sniffer) and that these commands also contain the new 802.11 network parameters.

Figure 4-6. Basic operation of the control script.

```
for background_load in “64” “128” “384” “512” … do # kbps
   for transmission_rate in “1” “2” “5.5” “11” do
      for transmission_power in “-30” “-31” “-32” “-33” … do # dBm
         for retry_limit in “0” “1” “2” … “8” do
            # ---- Start new transaction run ----
            # Set transmission rate and retry limit of card
            set_wlan_local(retry_limit, transmission_rate)
            # Start sniffer
            send_command_to(sniffer, “start”, bytes_per_packet)
            # Start SIP server stack
            send_command_to(access_point, “start”, transmission_power, retry_limit, transmission_rate)
            # Start traffic generator
            send_command_to(traffic_generator, “start”, background_load)
            # Start SIP client stack. Automatically exits when transaction run is done.
            start_client()
            # ---- Transaction run completed ----
         done
      done
   done
done
```

The SIP client initiates a series of transactions with the SIP server at random (but configurable) intervals. The SIP client and server log the (re)transmission properties of each transaction, for instance in terms of the local time at which a message arrived, how many 200 OKs were transmitted, and if the transaction succeeded or failed. *Figure 4-7* shows where we made these measurements in the SIP stack.

Figure 4-7. Measurement points in a SIP transaction.



Using the information provided by the 802.11 card, the SIP client also makes signal-to-noise ration (SNR) measurements and logs the average SNR during a transaction. The SIP client calculates the average SNR by measuring the instantaneous SNR when it (re)transmits/(re)receives certain

messages (e.g., INVITEs and 200 OKs). The SIP client and server store their transaction logs in memory in order not to affect the measurements through I/O operations. Multiple transactions may be in progress at the same time if the SIP client initiates a new transaction before the previous one has terminated, typically as a result of packet loss on the wireless link.

While the SIP client and server are executing transactions, the sniffer makes a network-level log of the messages it captured on the wireless link. We set the sniffer is in monitor mode, which means that it captures all the packets that traverse the wireless link, including 802.11 retransmissions. The sniffer dumps the packets it captured to file on-the-fly. The maximum size of the SIP messages we used is around 900 bytes, which means that they fit in a single UDP packet (i.e., no IP or 802.11 fragmentation).

When the SIP client has initiated the last transaction of a series, it waits 2*32 seconds for any ongoing transactions to finish. The reason for this dampening period is that the worst-case transaction requires 6 INVITE retransmissions (31.5 seconds) and 10 200 OK retransmissions (another 31.5 seconds).

After the dampening period, the SIP client transmits a stop command to the SIP server. At that point, both of them dump their logs to file and exit. This terminates the current iteration of the control script.

To avoid ARP delays, the control script inserts a manual entry for the access point in the SIP client's ARP cache before it enters the nested for-loop (not shown in *Figure 4-6*).

Analysis

Before analyzing the measurements, we first fed the logs of the SIP client, the SIP server, and the sniffer through an integration script. This script creates an integrated log by matching the client-side measurements of a specific transaction with the measurements of that transaction made at the server and at the sniffer. We then used Excel and various Visual Basic scripts to analyze the data.

4.4.3 Physical Environment

Figure 4-8 shows how we physically arranged the machines in the set-up. The distance between the SIP client (a laptop) and the SIP server (the access point/PC) is about 1.20 meters. The traffic generator is located about halfway between the SIP client and the SIP server. The sniffer is about 20 centimeters away from the SIP server, which enables it to capture packets sent by the SIP server (which may operate at a low transmission power) as well as packets sent by the SIP client (which operates at full power).

To create very low signal levels at the SIP client (i.e., 'put' it at the egde of the 802.11 cell), we had to wrap the access point's network card in foil (see the enlargement in *Figure 4-8*). The reason is that the card's firmware supports a lowest transmission power of -43 dBm, which still results in a good SNR at the SIP client.

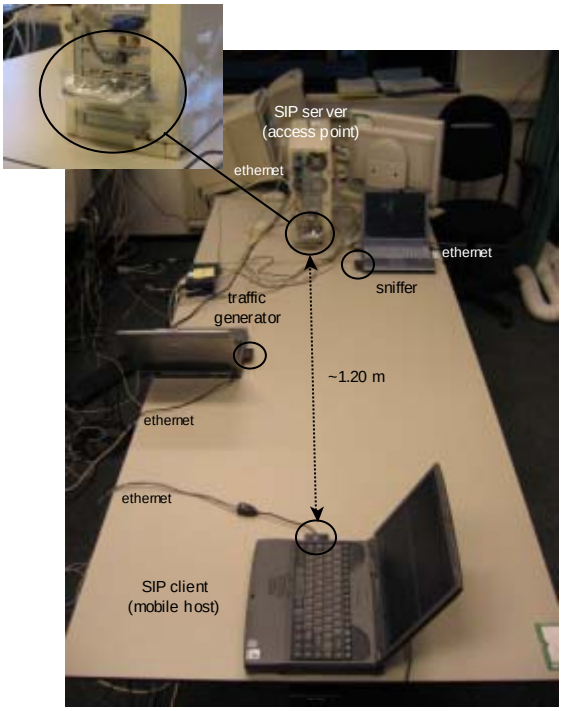


Figure 4-8. Physical arrangement of the measurement set-up.

The whole set-up is located in a computer lab. The 802.11 sniffing tool Kismet [kismet] showed no other networks on the channel that the access point uses (11). We configured other 802.11b networks under our control to make use of channels 1 through 7 to minimize interference. Networks that were not under our control already used channels in the 1-7 range. To further ensure that there were no interfering 802.11 networks, we also checked some of the sniffer's logs for foreign beacons. There turned out to be none, which indicates that the environment was indeed free from interference of other 802.11 networks.

The laptops in the set-up run on AC power to exclude influences from battery drain (e.g., lower processor speeds).

4.4.4 Hardware and Software Used

Table 4-1 provides an overview of the hard and software that we used in our set-up.

Table 4-1. Hard and software used in the measurement set-up.

	Device	802.11	Software
SIP client	Toshiba Satellite Pro 2400 laptop; 500 MHz Intel Pentium III Celeron processor; 192 MB RAM; Redhat Linux 2.4.20-8	Lynksys WPC11 PCMCIA card, version 3; hostap driver, version 0.2.4 (see http://hostap.epitest.fi)	Open SIP [OpenSIP], version 1, enhanced with measurement code
SIP server	PC, 2 GHz Intel Pentium IV; 1 GB RAM; Debian Linux 2.6.8.1	Lynksys WPC11 PCMCIA card, version 3; hostap driver, version 0.2.5 (see http://hostap.epitest.fi)	Open SIP [OpenSIP], version 1, enhanced with measurement code
Sniffer	Sony Vaio laptop; Redhat Linux 2.6.8; 1600 MHz Intel Pentium M processor; 1GB RAM	Orinoco Gold card, Lucent/Agere firmware version 6.06; driver patch 0.13e for monitoring mode support (see http://airsnort.shmoo.com/orinocoinfo.html)	tcpdump [tcpdump]; libcap 3.8.3, development version (October 7, 2004); tcdpump 3.8.3, development version (October 7, 2004) [tcpdump]
Traffic generator	Toshiba Tecra 8200 laptop; 750 MHz Intel Pentium III processor; 255 MB RAM; Redhat Linux 2.4.20-8	Orinoco Gold card, Lucent/Agere firmware version 8.72	jtg [jtg]
Control script	-	-	Bash/expect [expect]
Integration script	-	-	Bash; libcap 0.8.3; tcdpump 3.8.3, development version (September 12, 2004) [tcpdump]

4.5 Results

In this section, we consider the results of our experiments. The goal of these experiments is to determine how different 802.11 network parameters influence the retransmission behavior of a SIP transaction, specifically at the edges of an 802.11 cell (see Section 4.1). Since our measurement set-up is asymmetrical (see Section 4.4.2), we concentrate on the retransmission behavior of SIP transactions in terms of retransmitted 200 OKs.

Table 4-2 provides an overview of the 802.11 setting in the experiments we conducted. The 802.11 transmission rate is usually 1 Mbps, since this is typical rate for mobile hosts at the edge of an 802.11 cell. For the same reason, the SNR is usually at the 'low' end of the SNR spectrum, typically varying from around 0 to 17 dB. In all of these experiments, the SIP client randomly initiates a transaction either every second or every 2 seconds.

Table 4-2. Experiments overview.

Experiment	SNR	Transmission rate	Retry limit	Background	Sections
1,2	Low, fixed	1 Mbps	0	0 Mbps	4.5.1, 4.5.2
3	Low, variable	1 Mbps	0, 2, 4, 6, 8	0 Mbps	4.5.3
4	Low, variable	1 Mbps	0, 2, 4, 6, 8	1 Mbps	4.5.4
5	High, fixed	1 Mbps	0, 2, 4, 6, 8	variable	4.5.5
6	Low, variable	2, 5.5, 11 Mbps	0, 2, 4, 6, 8	0 Mbps	4.5.6

Note that the first two experiments merely serve to determine the length of a transaction run and to check that our measurement set-up is stable.

In Section 4.5.7, we will use the results of the above experiments to calculate the worst-case delays of the ALIVE protocol based on the formulas of Section 4.2.3.

4.5.1 Experiment 1: Length of a Transaction Run

The goal of our first experiment was to find an appropriate value for the length of the transaction runs executed by the SIP client and the SIP server (see Section 4.4.1). To accomplish this, we had to find the number of transactions in a run such that that the outcome of our measurements (e.g., in terms of the average number of transmitted 200 OKs per transaction) would not significantly change if we were to increase the number of transactions in the run.

To find this value, we executed a two test runs, A and B. Each run consisted of 5000 transactions, so that we could average our results over a run of m transactions ($1 \leq m \leq 5000$). We set the 802.11 network parameters to their worst case values (low transmission powers, no 802.11-level retries, and a 1 Mbps transmission rate) so that the outcome would also be applicable to experiments executed under better network conditions. The transmission power of the access point was –36 dBm during run A and –35 dBm during run B, which 'put' the SIP client at the edge of an 802.11 cell. As a result of this setting, the average SNR at the SIP client is lower during run A than during run B. We executed both runs without any background traffic.

Figure 4-9 shows the average number of transmitted 200 OKs per transaction for experiment A. The x-axis shows the number of transactions in a run (m) as multiples of 5 ($5 \leq m \leq 1500$). The line continues in the same manner for $1500 < m \leq 5000$ (not shown). The error bars indicate the standard deviation.

The curve in *Figure 4-9* shows that the average number of transmitted 200 OKs fluctuates until around $m = 500$. This suggests that runs of 500 transactions are appropriate. *Figure 4-9* also shows that the standard deviation stabilizes at around $m = 200$. The outcome of run B (not shown) is similar. In that case, the fluctuations in the average transmitted 200 OKs disappear at around $m = 300$.

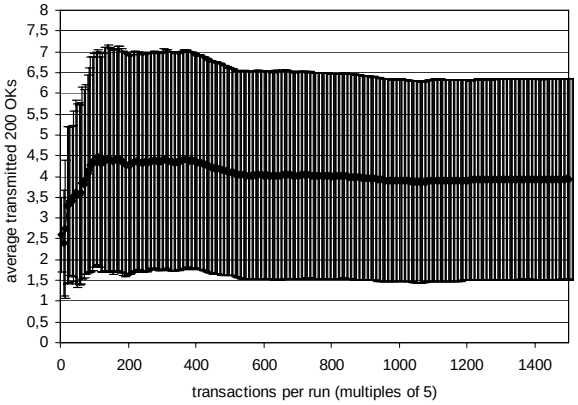


Figure 4-9. Average number of transmitted 200 OKs plus standard deviation during stability experiment A.

Figure 4-10 shows the average SNR per transaction for the first 1500 transactions of run A. The curve in *Figure 4-10* averages around 2.8 dB, which indicates that the run was executed under stable radio conditions.

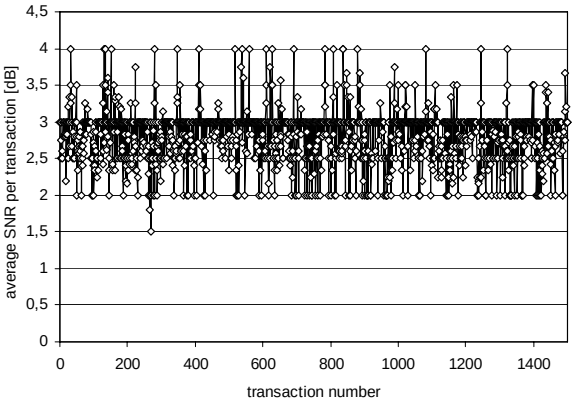


Figure 4-10. Average SNR per transaction for test run A.

4.5.2 Experiment 2: Stability of Set-up

The goal of our second experiment was to determine if our measurement set up is stable. We consider the set-up stable if the average delay between the transmission of the first INVITE and the reception of the corresponding 200 OK stabilizes. The idea is that the delay between an INVITE and a 200 OK includes the processing delay of the SIP server's stack, but not that of the SIP stack on the client. If the average delay does not stabilize (e.g., it continues to increase), then this is an indication that the server stack is behaving abnormally (e.g., it continues to build up transaction state).

Similarly, if the average delay between a 200 OK and an ACK on the server does not stabilize, then this is an indication that the client SIP stack is behaving abnormally. Observe that we configured both stacks such that they removed all of a transaction's state at the end of the transaction's maximum life-time (32 seconds after the reception of the first 200 OK for the client part of transaction, and 32 seconds after the transmission of the first 200 OK for server side transactions).

To check the stability of our set-up, we reused the data we gathered during runs A and B of Section 4.5.1. *Figure 4-11a* shows the average INVITE-200 OK delay and the average 200 OK-ACK delay per transaction for run A. The x-axis represents the number of transactions m in the run, again in multiples of 5 ($5 \leq m \leq 5000$). The average INVITE-200 OK delay stabilizes at around 6 seconds, whereas the average 200 OK-ACK delay stabilizes at around 5.8 seconds. The average 200 OK-ACK delay is probably somewhat higher because it includes the processing delay of the SIP client, which is a laptop that is less powerful than the SIP server (a PC). The results of experiment B (not shown) are similar, except that the average INVITE-200 OK delay stabilizes at around 1.7 seconds and the average 200 OK-ACK delay at around 1.9 seconds. *Figure 4-11*b shows that the standard deviation of run A stabilizes at around 7 seconds for the INVITE-200 OK delay, and at around 7.2 seconds for the 200 OK-ACK delay.

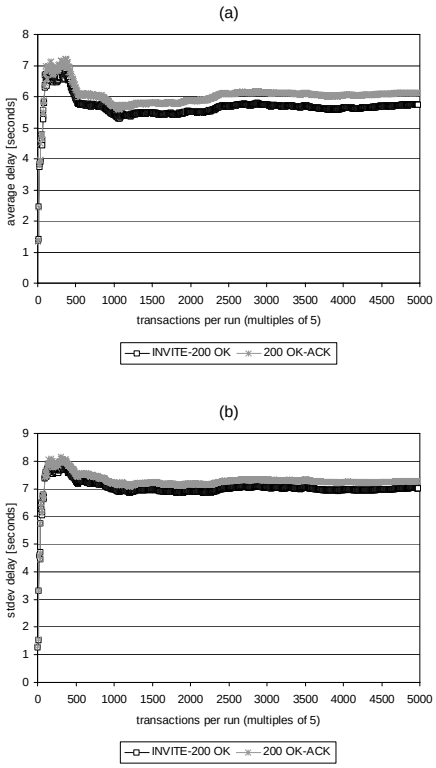


Figure 4-11. Average delays between an INVITE and a 200 OK and between a 200 OK and an ACK for transaction run A (a) and the associated standard deviation.

For completeness, we also measured the processing delay of the SIP stack on the SIP client and on the SIP server. *Figure 4-12a* shows the average delay per client transaction to transmit an INVITE and handle an incoming 200 OK; *Figure 4-12b* shows the average time for a server transaction to handle an incoming INVITE and to transmit a 200 OK. The x-axis represents the number of transactions in a run in multiples of five.

All four curves in *Figure 4-12* stabilize quickly, which further suggests that the entire set-up is stable. *Figure 4-12b* also shows that the processing delays of the SIP stack on the server is neglectable. *Figure 4-12a* shows that the average processing delay on the client (around 4.5 milliseconds) is non-neglectable, but that it is stable.

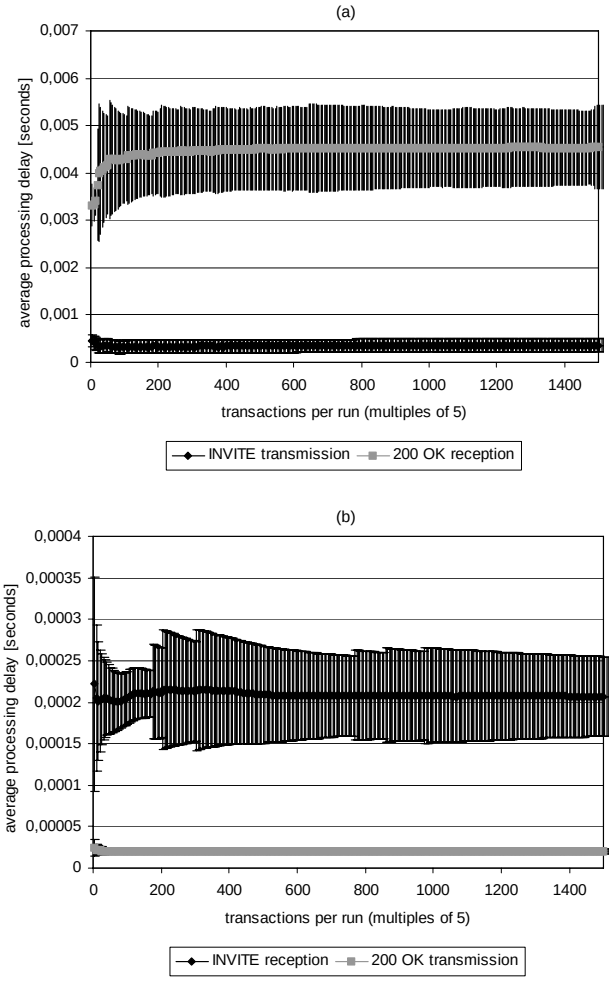


Figure 4-12. Client-side SIP processing delays (a) and server-side processing delays (b) for run A.

4.5.3 Experiment 3: At the Edge of an Unloaded 802.11 Network

The goal of experiment 3 is to analyze the retransmission behavior of SIP transactions at the edge of an 802.11 cell. We measure the percentage of transactions that require retransmissions for various SNRs and 802.11 retry limits as well as the associated SIP back-off delay. The 802.11 transmission rate in this experiment is 1 Mbps and the network does not carry any competing traffic.

Experiment 3 is based on 145 transaction runs, each consisting of 500 transactions. The transaction runs took place at 29 different SNR levels between 0 and approximately 17 dB and using five different 802.11 retry limits (0, 2, 4, 6, and 8).

Transmitted 200 OKs

Figure 4-13 shows the relative number of 200 OK transmissions for the 145 transaction runs of experiment 3. We plot this as a function *F(t, r, s)*, which represents the percentage of successful transactions that required t 200 OK transmissions in the transaction run at SNR level s (in dB) and at retry limit r. *Figure 4-13*a through *Figure 4-13*e shows F(t, r, s) for the five retry limits we consider in this experiment (0, 2, 4, 6, and 8).

While the number of 200 OK transmissions in a transaction lies in the range 1-11 (see Section 4.3.1), we only plot F(t, r, s) for the range 1 through 5 to increase the readability of *Figure 4-13*. The graphs in *Figure 4-13* also show *E(r, s)*, which is the number of transactions that failed at a particular retry limit r and SNR s.

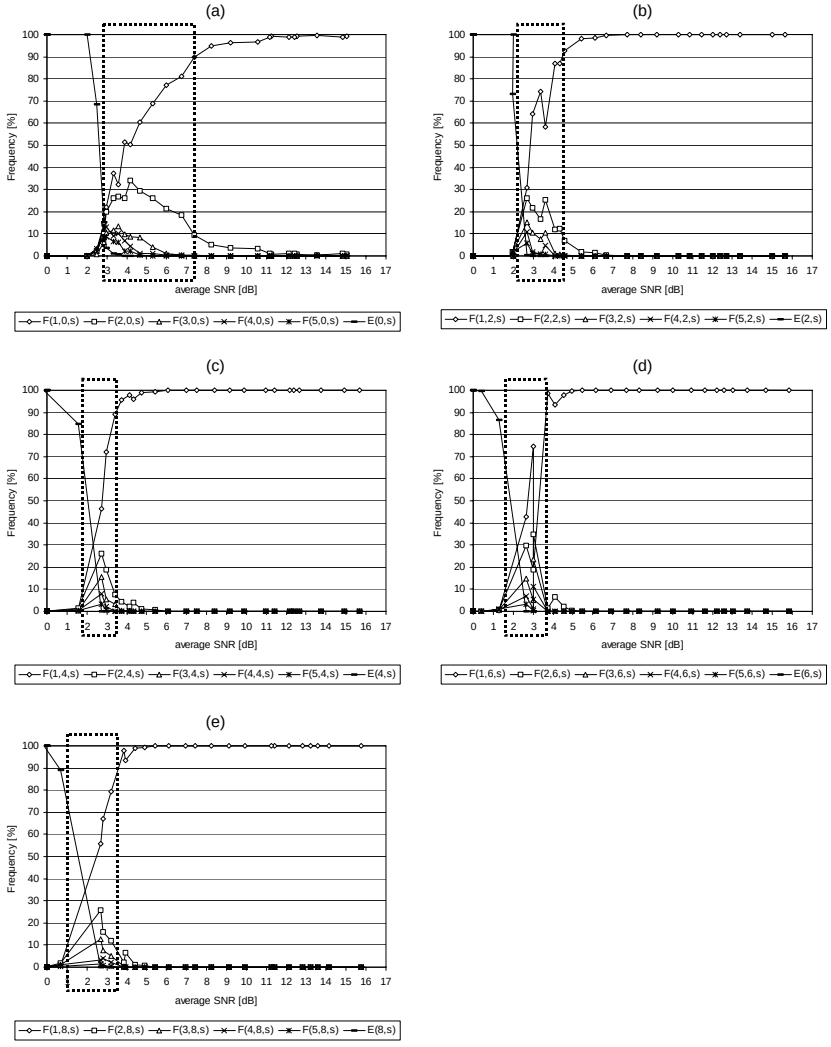


Figure 4-13. Transmission behavior of SIP transactions. The curves indicate the percentage of SIP transactions that required t 200 OK transmissions ($1 \leq t \leq 5$). The variables are the SNR and the retry limit (0, 2, 4, 6, and 8 in graphs a, b, c, d, and e, respectively). The constants are the transmission rate is (1 Mbps) and the background load (0 Mbps).

As expected, *Figure 4-13* shows that an increased 802.11 retry limit increases the percentage of SIP transactions that only need to transmit one 200 OK. For example, without any retries, 65% of the transactions requires only one 200 OK transmission at 5 dB (i.e., F(1, 0, 5) = 65%). With 2 or more retries, almost 100% of the transactions requires only one 200 OK transmission at 5 dB (e.g., F(1, 2, 5) equals around 95%). *Figure 4-13* also shows that an increase in the retry limit enables SIP transactions to deliver 200 OKs in one transmission at lower SNRs. For example, without retries, the SNR must be around 7.4 dB for 90% of the SIP transactions to deliver a 200 OK in one shot (i.e., F(1, 0, 7.4) = 90%). For 2 retries this is at approximately 4.4 dB and for 4, 6, and 8 retries it is around 3.5 dB.

Fall-off Regions

The dotted rectangles in *Figure 4-13* represent what we call *fall-off regions*, which are the dB ranges in which F(1, r, s) falls from 90% to 10% (cf. [Aguayo04]). Within a fall-off region, the sum of the other F(t, r, s)'s ($2 \leq t \leq 11$) plus E(r, s) increases from 10% to 90%.

Figure 4-13 shows that 802.11 retransmissions decrease the width of the fall-off region. For example, the fall-off region is 4.6 dB wide without any 802.11 retries (*Figure 4-13a*) and around 1.7 to 2.5 dB wide with retries (*Figure 4-13b-e*). The advantage of a small fall-off region is that the number of SIP retransmissions will remain relatively constant when the SNR decreases. Only in the fall-off region (i.e., at the edge of the network) will the number of retransmissions increase sharply.

Although we only used five retry limits, *Figure 4-13* suggests that the width of the fall-off region does not significantly decrease beyond a retry limit of 2. For 2, 4, 6, and 8 retries, the width of the fall-off regions is around 2.2 dB, 1.7 dB, 2.1 dB, and 2.5 dB, respectively.

To see if a retry limit higher than 2 would be beneficial inside the fall-off region, we calculated the average percentage of 200 OK transmissions (i.e., the average of the values of F(t, r, s)) inside the fall-off regions for the retry limits of 2, 4, 6, and 8. *Figure 4-14* shows the result. The y-axis of *Figure 4-14* indicates the average of F(t, r, s) in the fall-off region, while the x-axis specifies the number of 200 OK transmissions, t.

Figure 4-14 suggests that an increase in the retry limit to 4, 6, or 8 only marginally improves the average percentage of transactions in the fall-off region that manage to deliver their 200 OK in one shot (i.e., the average of F(t, r, s) at $t=1$ does not significantly increase). Equivalently, increasing the retry limit to 4, 6, or 8 does not significantly reduce the percentage of transactions that requires one or more retransmissions to deliver a 200 OK (i.e., the average of F(t, r, s) at a particular $t \geq 2$ does not significantly decrease). The reason is probably that the quality of the connection in the fall-off region is so bad that the additional 802.11 retries are lost as well.

Based on this information, we draw the preliminary conclusion that a retry limit of 2 suffices for SIP-based applications like the ALIVE protocol, at least in an environment without interference and background traffic. This may be advantageous for packets carrying the actual multimedia data because a smaller retry limit can reduce the time these packets spend in the MAC queue waiting for the (re)transmission of packets at the head of the queue [Punnoose01].

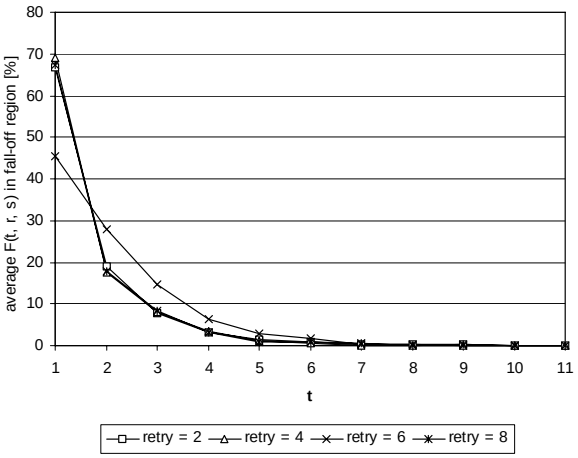


Figure 4-14. Average number of 200 OK transmissions in the fall-off region for different retry limits.

Notice that the transaction run at 3.03 dB with a retry limit of 6 (*Figure 4-13d*) experienced an unusual amount of packet loss, perhaps because of interference of some sort.

As of around 3 dB, the percentage of failed transactions (i.e., E(r, s)) dramatically for all five retry limits. 90% of the transactions fail at around 2 dB (retry limits zero and two) or at around 1 dB (the other three retry limits).

Back-off Delays

In experiment 3, the delay between the transmission of a transaction's first INVITE and the arrival of the first 200 OK is mainly determined by the number of 200 OKs transmitted during the transaction. As a result, the INVITE-200 OK delay should follow the delays of the back-off sequence for retransmitting 200 OKs (i.e., 0.5 seconds for one retransmission, 1.5 seconds for two retransmissions, 3.5 seconds for three retransmissions, and so on).

To check this, we plotted *D(d, r, s)*, which is the percentage of transactions with an INVITE-200 OK delay in the interval [d, d+0.25) seconds in a transaction run with retry limit r and SNR s. D(d, r, s) should be about the same as F(t, r, s) if d is the back-off delay for t transmissions. For example, D(0.5, r, s) should be about the same as F(2, r, s) since the second transmission (the first retransmission) of a 200 OK occurs 0.5 seconds after the transmission of the original 200 OK. Similarly, D(1.5, r, s) should be about the same as F(3, r, s), D(3.5, r, s) about the same as F(4, r, s), and so on.

Figure 4-15 plots D(d, r, s) for the retry limits 0 (*Figure 4-15a*) and 8 (*Figure 4-15b*), and shows that D(d, r, s) indeed largely follows the same

curve as F(t, r, s) for the corresponding retry limits (see *Figure 4-13a* and *Figure 4-13e*).

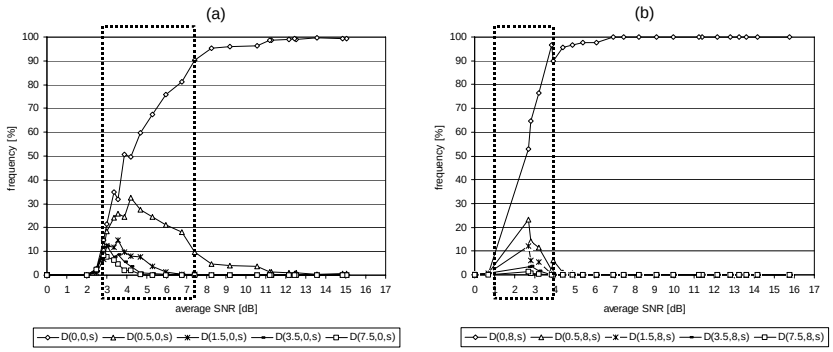


Figure 4-15. Percentage of SIP transactions that incurs d seconds of delay between an INVITE and a 200 OK using different SNR levels and retry limits of 0 (a), and 8 (b).The transmission rate is 1 Mbps and the network is unloaded.

4.5.4 Experiment 4: At the Edge of a Saturated 802.11 Network

The goal of experiment 4 is to analyze the retransmission behavior of SIP transactions at the edge of an 802.11 cell that is fully loaded with competing traffic. As in experiment 3, we measure the percentage of retransmissions for various SNRs and retry limits as well as the associated back-off delay. The 802.11 transmission rate in this experiment is 1 Mbps and the network carries 1 Mbps of UDP Constant Bit Rate (CBR) background traffic. The packet size of the background traffic is 1000 bytes.

Stability

Since the background traffic might increase packet loss on the network, we first checked if 500 transactions per run still yielded stable results (cf. test runs A and B of Section 4.5.1).

Figure 4-16 shows the results of the test run. *Figure 4-16a* shows the average number of transmitted 200 OKs over m transactions, where m is a multiple of 5 and $5 \leq m \leq 500$. The average number of transmitted 200 OKs stabilizes at around 3.8, which is at approximately 300 transactions. The error bars in *Figure 4-16a* indicate the standard deviation of the average transmitted 200 OKs. The bars show that the standard deviation of the transmitted 200 OKs ends up at around 2.7.

The average delay between an INVITE and a 200 OK and the average delay between a 200 OK and an ACK (*Figure 4-16b*) also stabilize near 300 transactions. The same holds for their standard deviations (*Figure 4-16c*). We therefore conclude that 500 SIP transactions is still a good value for the length of a transaction run if the network is saturated.

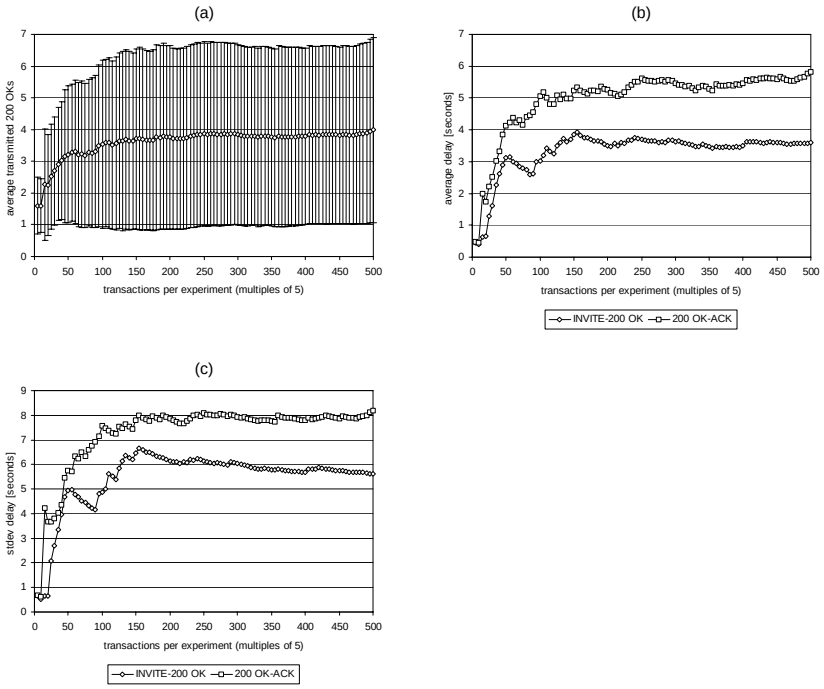


Figure 4-16. Stability run for a saturated network. Average and standard deviation of the number of transmitted 200 OKs (a), average delay between an INVITE and a 200 OK and between a 200 OK and an ACK (b) and the associated standard deviations (c).

Fall-off Regions

Experiment 4 is based on 140 transaction runs, each consisting of 500 transactions. The transaction runs took place at 28 different SNR levels between 0 and approximately 14 dB and using five different 802.11 retry limits (0, 2, 4, 6, and 8). *Figure 4-17* shows the relative number of 200 OK transmissions of the transactions in the runs, again organized according to retry limit (0 retry limit in *Figure 4-17a*, a retry limit of 8 in *Figure 4-17e*).

Figure 4-17 shows the same pattern as *Figure 4-13*, specifically that the width of the fall-off region decreases as the 802.11 retry limit increases. In addition, the width of the fall-off regions seem to be similar to those in the unloaded network. The width of the fall-off region is about 3 dB for a retry limit of 2 (*Figure 4-17b*, 2.2 dB in *Figure 4-13b*), 2 dB with a retry limit of 4 (*Figure 4-17c*, 1.7 dB in *Figure 4-13c*), 2.3 dB for a retry limit of 6 (*Figure 4-17d*, 2.1 dB in *Figure 4-13d*), and 1.8 dB for a retry limit of 8 (*Figure 4-17e*, 2.5 dB in *Figure 4-13e*). This suggests that 802.11 MACs divide the link bandwidth fairly amongst the mobile hosts in a cell. The only exception seems to be the case where the 802.11 network does not retransmit frames, in which case the width of the fall-off region is around 11 dB (*Figure 4-17a*, 4.6 dB in *Figure 4-13a*).

Notice that the width of the fall-off region in *Figure 4-17a* is an estimate because F(1, 0, s) did reach 90% in experiment 4. Similarly, the width of

the fall-off region in *Figure 4-17b* is also an estimation because F(1, 2, s) did not cross the 10% threshold.

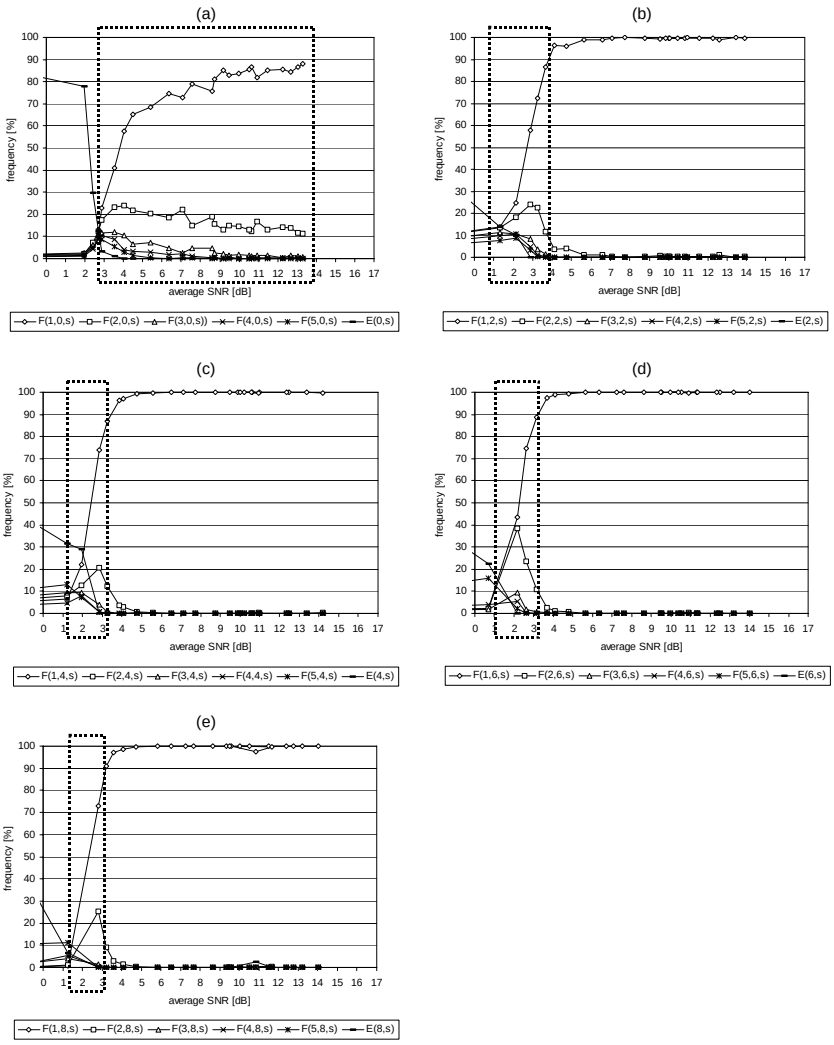


Figure 4-17. Transmission behavior of SIP transactions. The curves indicate the percentage of SIP transactions that required t 200 OK transmissions ($1 \leq t \leq 5$). The variables are the SNR and the retry limit (0, 2, 4, 6, and 8 in graphs a, b, c, d, and e, respectively). The constants are the transmission rate is (1 Mbps) and the background load (1 Mbps CBR traffic).

Figure 4-18 plots the average of F(t, r, s) inside the fall-off regions of *Figure 4-17*. It shows that the average percentage of transactions that require one 200 OK increases as the retry limit increases and that the difference between a retry limit of 2 and a retry limit of 8 is around 22%. The difference between the average percentage of transactions that transmit two or more 200 OKs gets smaller when the retry limit increases. The difference is at most 10% (at $t = 2$).

From *Figure 4-18* we observe that a retry limit higher than 2 slightly improves the percentage of transactions that deliver their 200 OK in one shot, but that it does not significantly reduce the percentage of transactions that require two or more 200 OK transmissions. A retry limit of 4 or perhaps 6 therefore seems appropriate compared to the case where the network is unloaded (see Section 4.5.3).

We note that the number of transaction runs executed inside the fall-off region decreases as the retry limit increases. As a result, the average in *Figure 4-18* are based on only a few transaction runs, in one case even on only one (retry limit of 8).

Figure 4-18. Average percentage of transactions with t 200 OK transmissions inside the fall-off region.

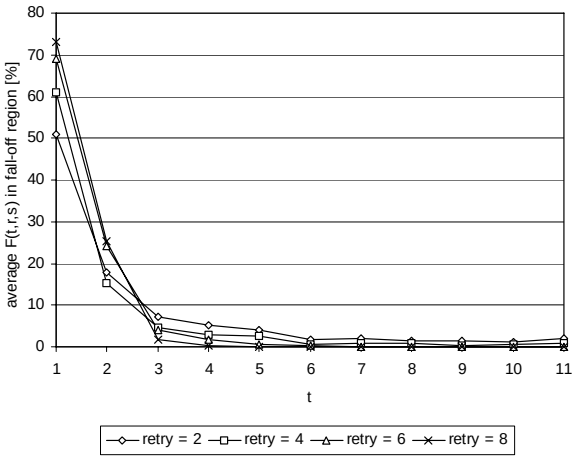
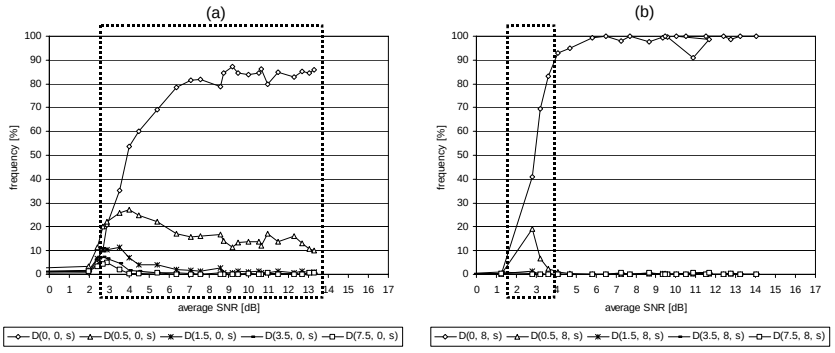


Figure 4-19 shows D(d, r, s) (see Section 4.5.3) using the retry limits 0 and 8. *Figure 4-19* shows that D(d, 0, s) and D(d, 8, s) largely follow the delays associated with a 200 OK's back-off sequence (*Figure 4-19a* and *Figure 4-19b*).

Figure 4-19. Relative frequency of INVITE-200 OK delays in a fully loaded network.



4.5.5 Experiment 5: Variably Loaded Network

The goal of experiment 5 is to determine the effects of background traffic on the behavior of SIP transactions. We therefore configured the SNR to a 'good' value (around 32 dB on average) and ran 45 transaction runs using 9 different CBR background loads and 5 different retry limits.

Transmitted 200 OKs

For experiment 5, function F (see Section 4.5.3) has the form F(t, r, b), where t is a number of 200 OK transmissions, r is a retry limit, and b is a background load (kbps). The SNR does not appear in F because it is a constant in experiment 5.

Figure 4-20 plots the retransmission behavior of the SIP transactions at 9 bandwidth levels (0…1024 kbps, multiples of 128 kbps) without any 802.11 retries. *Figure 4-20* indicates that the percentage of transactions that requires one or more retransmissions only increases at high background loads (as of 640 kbps) and that the maximum is 12.4% (see F(2, 0, 896)). This suggests that under 'good' radio conditions a CBR background load has little effect on the number of 200 OK retransmissions, even if the retry limit is 0. The number of transactions that retransmit a 200 OK is virtually 0 for a retry limit of 2, 4, 6, or 8 (not shown).

From *Figure 4-20* we can conclude that CBR background traffic requires few SIP transactions to retransmit a 200 OK if the SNR is 'good'. A retry limit of 2 suffices to reduce the number of 200 OK retransmissions to virtually zero.

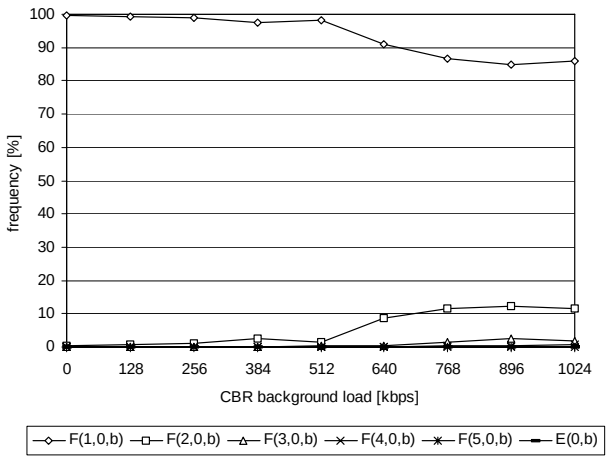


Figure 4-20. Transmission behavior of SIP transactions. The curves indicate the percentage of SIP transactions that required t 200 OK transmissions ($1 \leq t \leq 5$). The variables are the amount of background traffic. The constants are the retry limit (0), the transmission rate (1 Mbps) and the SNR (around 32 dB on average).

4.5.6 Experiment 6: Unloaded Network, Different Rates

In experiment 6, we experimented with SIP transactions using the three other 802.11 transmission rates (2, 5.5, and 11 Mbps). For each of these transmission rates, we executed 120 transaction runs (500 transactions each), divided over 24 SNR levels and 5 retry limits. In this experiment, the network was unloaded.

Fall-off Regions

Figure 4-21 shows the fall-off regions at the three different transmission rates (columns) and five retry limits (rows). We denote the percentage of SIP transactions that transmitted a 200 OK t times as F(t, r, s, tx), where r is a retry limit, s an SNR value, and tx a transmission rate. From left to right, *Figure 4-21* plots F(t, r, s, 2), F(t, r, s, 5.5), and F(t, r, s, 11). From top to bottom, it plots F(t, 0, s, tx), F(t, 2, s, tx), F(t, 4, s, tx), F(t, 6, s, tx), and F(t, 8, s, tx).

Figure 4-21 shows that the fall-off regions shift to the right as the transmission rate increases (i.e., left to right). For example, with a retry limit of 2 and a transmission rate of 2 Mbps, the fall-off region begins at approximately 3.8 dB (*Figure 4-21*d). With the same retry limit and a transmission rate of 5.5 dB (*Figure 4-21*e), the fall-off region begins at 5.6 dB, while at 11 Mbps it begins at around 9.8 dB (*Figure 4-21*f). These measurements confirm that higher bitrate modulation schemes are more sensitive to packet loss than lower rate ones [Hoene03, Agulayo04].

Figure 4-21 also shows that the width of the fall-off region decreases as the retry limit increases (cf. Section 4.5.3 for a 1 Mbps transmission rate). For example, at 11 Mbps (third column in *Figure 4-21*) the width of the fall-off region is approximately 3.5 dB, 2.2 dB, 1.5 dB, 1.5 dB, and 1.4 dB with a retry limit of 0, 2, 4, 6, and 8, respectively.

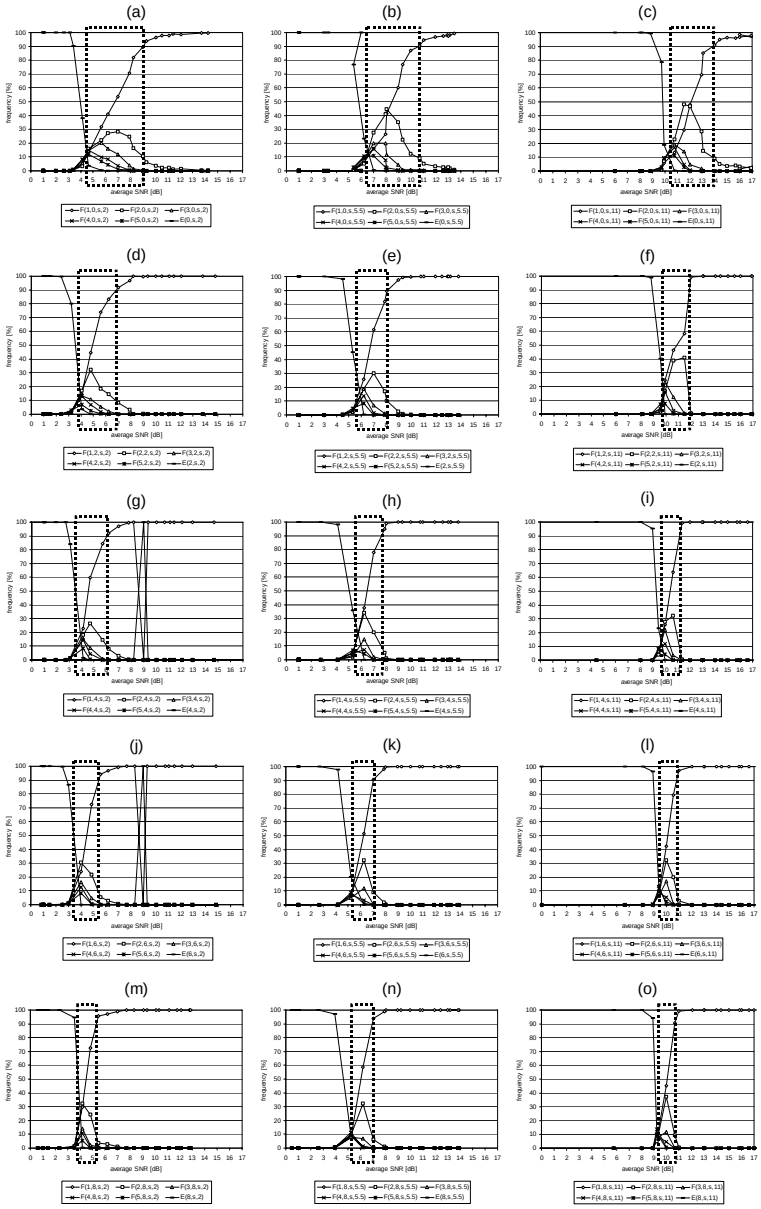


Figure 4-21. F(t, r, s, tx) at 2 Mbps (left column), at 5.5 Mbps (middle column), and at 11mbps (right column). Each row represents one retry limit (0, 2, 4, 6, and 8).

Rate Adaptation

When a mobile hosts leaves a hotspot, it typically shifts down from 11 Mbps, to 5.5, to 2, and eventually to 1 Mbps. Conversely, the host will shift back up to 11 Mbps when it enters a hotspot. This behavior is called rate adaptation and should take place automatically [Haratcherev04]. Rate

adaptation is not part of the 802.11b standard, which means that it can also be controlled by applications, in this case the ALIVE system.

To avoid high delays, the mobile hosts in the ALIVE system should move to a lower rate before they end up in the fall-off region of their current transmission rate. For example, when a mobile host is using a retry limit of two and its current transmission rate is 11 Mbps, then it should shift to 5.5 Mbps before the SNR drops below 12 dB (see *Figure 4-21*f).

To identify at which moments a SIP-based application should shift to another rate, we plotted the fall-off regions of the four 802.11b rates in *Figure 4-22*. Each of the five graphs in *Figure 4-22* represent one specific retry limit (0, 2, 4, 6, or 8). Observe that *Figure 4-22* only plots F(1, r, s, tx) to keep it readable.

From *Figure 4-22*, we observe that the fall of regions of different transmission rates begin to overlap when the retry limit decreases. For example, at a retry limit of 8, the four fall-off regions are almost non-overlapping (*Figure 4-22e*), while there is a considerable overlap if the retry limit is 0 (*Figure 4-22a*). To avoid ending up in a fall-off region, this means that a mobile host will need to change its rate more often across the same dB range at lower retry limits than at higher ones.

Figure 4-22 thus provides a metric for the edge of an 802.11 cell, which depends on the SNR, the retry limit, and the transmission rate. This information can be used by switching controllers on mobile hosts to initiate configuration discovery.

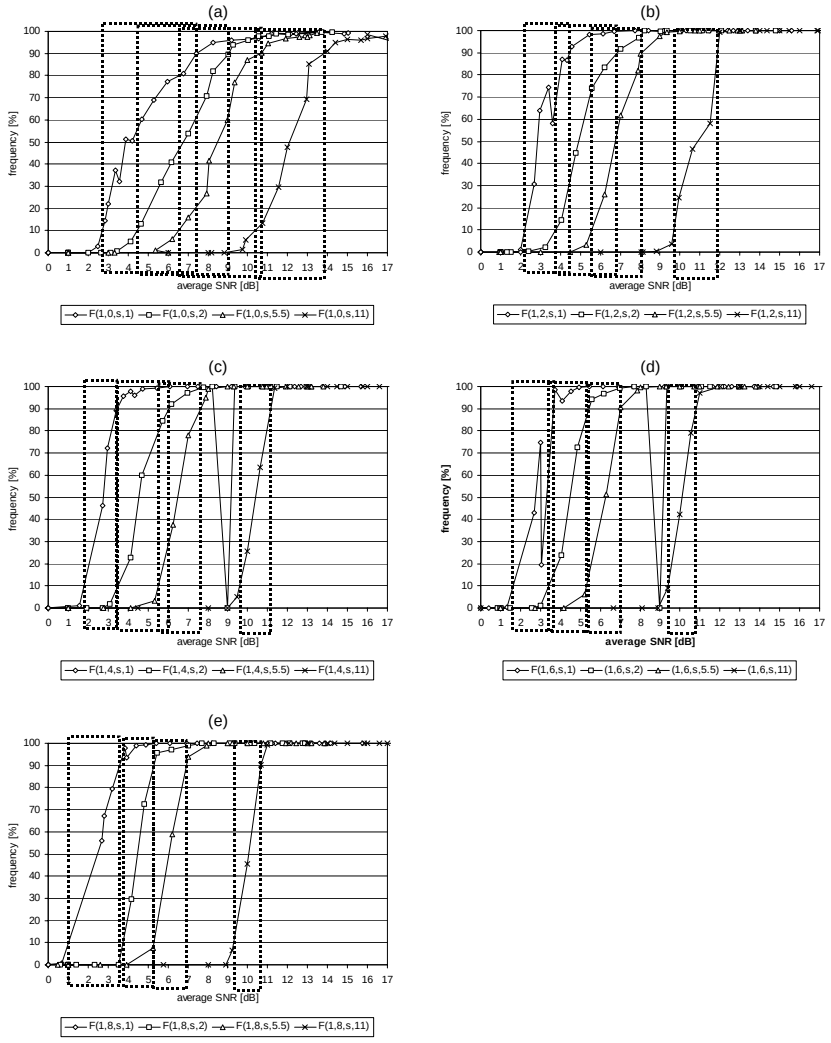


Figure 4-22. Fall-off regions at different transmission rates (1, 2, 5.5, and 11 Mbps) with retry limits of 0 (a), 2 (b), 4 (c), 6 (d), and 8 (e).

Fall-off Regions

Figure 4-23 shows the average value of F in the fall-off region for 2 Mbps (*Figure 4-23a*), 5.5 Mbps (*Figure 4-23b*), and 11 Mbps (*Figure 4-23c*). Similar to the 1 Mbps case (*Figure 4-14*), *Figure 4-23* suggests that a retry limit of more than 2 does not significantly reduce the percentage of transactions that requires one or more retransmissions to deliver a 200 OK in the fall-off region. Note however that the number of transaction runs that take place in the fall-off region decreases as the width of the fall-off region decreases. As a result, some of the averages in *Figure 4-23* are based on two or sometimes even one transaction run.

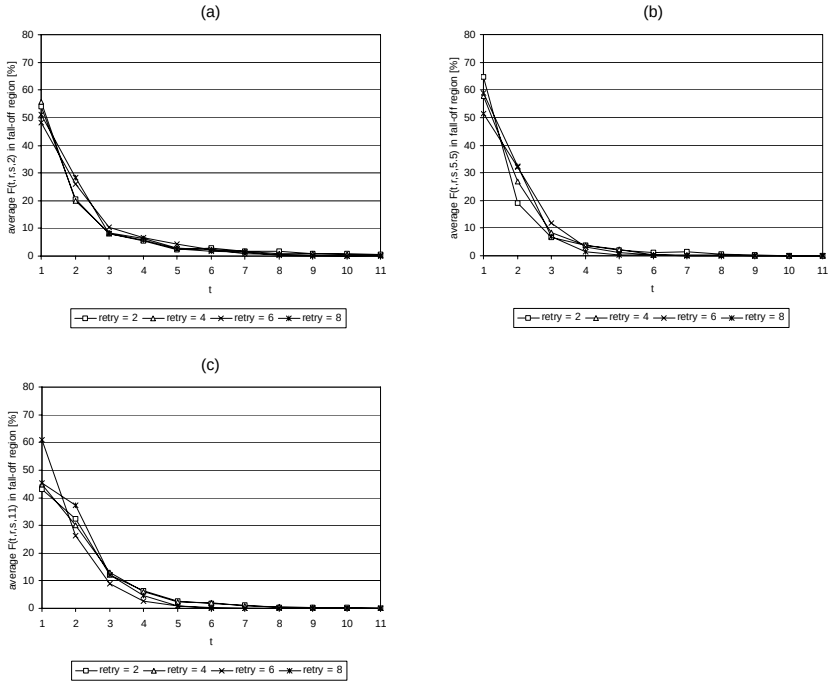


Figure 4-23. Average of F(t, r, s, tx) in the fall-off region for 2 Mbps (a), 5.5 Mbps (b), and 11 Mbps (c).

4.5.7 ALIVE Protocol Overhead Revisited

Using the data of experiment 3 (unloaded network, Section 4.5.3) and experiment 4 (saturated network, Section 4.5.4), we can now also provide worst-case values for the delay formulas of Section 4.2.3 (ALIVE protocol overhead). In this chapter, we use the average delay between an INVITE and a 200 OK for this purpose. We consider the average value in the middle of the fall-off region to be the worst one.

Unloaded Network (Experiment 3)

Figure 4-24 shows the average delays between an INVITE and a 200 OK in experiment 3 using a retry limit of 2. *Figure 4-24*a shows the average delay over all SIP transactions in a run whereas *Figure 4-24*b shows the average delay for the transactions in a run that do not involve any SIP retransmissions (i.e., their delays are close to the round-trip delay to traverse the wireless link). The dashed rectangles in *Figure 4-24* represent the fall-off region of *Figure 4-13*b. The vertical dashed line indicates the middle of the fall-off region.

The worst-case average delay over all SIP transactions (*Figure 4-24*a) is approximately 324 milliseconds (at around 3.3 dB). Since we used an asymmetrical measurement set-up in which the client always transmits at

full power (see Section 4.4), it is typically the path from the SIP server to the SIP client that is subject to SIP retransmissions. The one-way delay on this path is 324 milliseconds minus the one-way delay from the SIP client to the SIP server. The latter is equal to half the INVITE-200 OK delay of a SIP transaction that does not involve any SIP retransmissions (assuming symmetrical up and downlinks). *Figure 4-24*b shows that the worst-case average delay for such transactions is around 50 milliseconds (at around 3.3 dB), which means that the worst-case one-way delay from the SIP server to the SIP client is 324 – (50/2) = 299 milliseconds. In the formulas of Section 4.2.2, the one-way delay from a front-end and a mobile host (i.e., d(accessPoint, host) + d_{rtx}(frontEnd, sip)) and the one-way delay from the target media server to the mobile host (i.e., d(accessPoint, host) + d_{rtx}(targetMediaServer, sip)) therefore both equal 299 milliseconds when the 802.11 network is unloaded. For simplicity, we assume that the one-way delays in the other direction (host to front-end and host to target media server) are the same.

Substituting the 299 milliseconds in the formulas of Section 4.2.2, we get a worst-case value for the configuration query delay (t_{query}) of:

$$t_{query} = 4*299 + 1216 = 2412 \text{ milliseconds}$$

The worst-case switching delay (t_{switch}) in this case equals

$$2*299 + 20 = 618 \text{ milliseconds}$$

The resulting total delay of the ALIVE protocol then equals

$$t_{query} + t_{switch} = 3030 \text{ milliseconds}$$

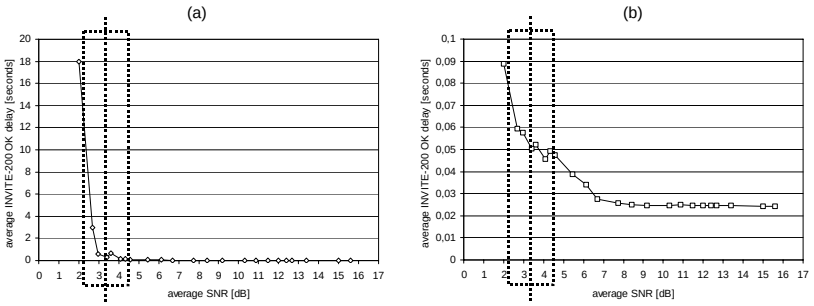


Figure 4-24. Average INVITE-200 OK delay in an unloaded network using a retry limit of 2. Graph (a) shows the average delays for all SIP transactions, whereas (b) shows the average delays for transactions without any SIP retransmissions.

Loaded Network (Experiment 4)

Figure 4-25 shows the average delays between an INVITE and a 200 OK in experiment 4 (network saturated with 1 Mbps of CBR background traffic)

using a retry limit of 4. Again, *Figure 4-25*a shows the average over all SIP transactions in a run whereas *Figure 4-25*b shows the average delay for the transactions without SIP retransmissions. The dashed rectangles represent the fall-off region of *Figure 4-17*c and the vertical dashed line the middle of the fall-off region.

The worst-case average delay over all SIP transactions (*Figure 4-25*a) is approximately 2500 milliseconds (at around 2.4 dB). Using the same rationale as for the unloaded network (asymmetrical set-up), the one-way delay from the SIP server to the SIP client equals 2500 milliseconds minus the one-way delay from the SIP client to the SIP server. Since 1 Mbps of CBR background traffic does not cause any SIP retransmissions using a 802.11 retry limit of 4 and a 'good' SNR value (see experiment 5, Section 4.5.5), there will be no SIP retransmissions as a result of packet loss on the path from the SIP client to the SIP server. The one-way delay from the SIP client to the SIP server is therefore half the INVITE-200 OK delay of a SIP transaction that does not involve any SIP retransmissions (assuming symmetrical up and downlinks). *Figure 4-25*b shows that the worst-case average delay for such transactions is around 240 milliseconds (at around 2.4 dB), which means that the one-way delay from the SIP server to the SIP client is 2500 – (240/2) = 2380 milliseconds. In the formulas of Section 4.2.2, d(accessPoint, host) + d_{rtx}(frontEnd, sip) and d(accessPoint, host) + d_{rtx}(targetMediaServer, sip) therefore both equal 2380 milliseconds. For simplicity, we again assume that the one-way delays in the other direction are the same.

Substituting the 2380 milliseconds in the formulas of Section 4.2.2, we get a worst-case value for the configuration query delay (t_{query}) of:

$$t_{query} = 4*2380 + 1216 = 10736 \text{ milliseconds}$$

The worst-case switching delay (t_{switch}) equals:

$$t_{switch} = 2*2380 + 20 = 4780 \text{ milliseconds}$$

The total worst-case delay of the ALIVE protocol ($t_{query} + t_{switch}$) equals:

$$t_{query} + t_{switch} = 15516 \text{ milliseconds}$$

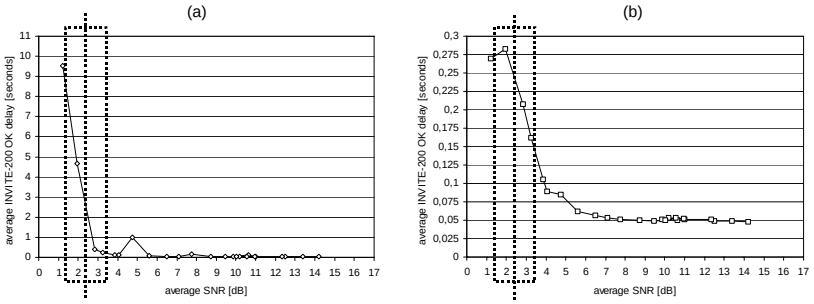


Figure 4-25. Average INVITE-200 OK delay in a network loaded with CBR traffic using a retry limit of 4. Graph (a) shows the average delays for all SIP transactions, whereas (b) shows the average delays for transactions without any SIP retransmissions.

From the above analysis we conclude that the playout buffer on a mobile host in the ALIVE system should be able to buffer about 15.5 seconds of multimedia information to deal with switches that takes place under the worst circumstances. Buffers of such a size are in line with the amount of buffering used by a contemporary media player such as RealPlayer [Li02]. However, such amounts of buffer space are usually only required in a small small dB region at the very edge of a cell, a point at which it will probably also be difficult to execute an IP handoff. Outside this small region, the ALIVE protocol will typically introduce a delay that is close to its minum value. This is around 56 milliseconds for an unloaded network (see Section 4.2.3) and about 108 milliseconds for a network loaded with 1 Mbps of CBR background traffic (the minimum average one-way delay in a loaded network is around 25 milliseconds, see *Figure 4-25*b).

4.6 Related Work

As far as we know, a thorough empirical investigation into the relation between SIP retransmissions and 802.11 packet loss parameters has not been reported before. Camarillo, Kantola, and Schulzrinne share this observation by pointing out that “more work is needed to study interactions between lossy links such as some radio interfaces and transport-layer or application-layer retransmissions” [Camarillo03].

Simulation and analytical approaches do however exist. Curcio et al. [Curcio01] used a SIP emulator to measure the delay introduced by SIP transaction over lossy wireless LAN links. They focused on telephony applications and measured the delay between an INVITE and a 180 Ringing (the post dialing delay). They vary radio coverage levels (percentages) and the packet loss rate. The exact meaning of a radio coverage level is however unclear plus that they do not consider the 802.11 parameters that influence their packet loss rate. Banerjee et al. [Banerjee04] take an analytical approach and use queuing models to analyze the one-way delay to transmit an INVITE from a mobile host to a correspondent host over an 802.11

link. They vary the frame error rate and the transmission rate of an 802.11 link as well as the rate at which a mobile host transmits INVITEs.

The delay introduced by SIP transactions has also been analyzed for UMTS networks [Banerjee03, Banerjee04, Curcio02] (analytically and through simulations), for UMTS satellite links [Kueh04] (simulations), and for the fixed portion of the Internet [Eyers00] (simulations).

The influence of 802.11 packet loss parameters has been empirically studied in other application areas, specifically in the context of multi-hop ad-hoc networks [Aguayo04] and the actual streaming of multimedia packets [Hoene03]. Their experiments are similar to ours and our results seem to be in line with theirs.

Chapter 5

Conclusions

We started this thesis with some observations on the complexity in the value chain from multimedia content providers ('sources') to mobile Internet users. The first challenge was to deliver live and scheduled multimedia content to a large number of heterogeneous receivers. We tackled this challenge in the design of the ALIVE business network, specifically by means of the introduction of aggregators, which form a ***second tier*** *between users and sources. Aggregators deal with heterogeneous receivers by transmitting channels in multiple* ***configurations****. A configuration consists of a set of streams with specific packetization and compression parameters that carry the content of a channel. The second challenge was to enable mobile users to seamlessly receive a channel while they roam across aggregators and access networks. We addressed this challenge in the ALIVE business network in the form of roaming agreements between aggregators, as well as in the design of the* ***ALIVE system****, in which appropriate configurations are* ***negotiated*** *between aggregators and users. We also addressed the challenge of seamless mobility in our analysis, where we investigated under which 802.11 conditions the implementation of the ALIVE protocol on top of SIP yields acceptable delays.*

In this chapter, we consider the results of our work on the ALIVE business network and the design and analysis of the ALIVE system in more detail (Section 5.1). We also make the contributions of this thesis explicit (Section 5.2). We conclude with an overview of future work (Section 5.3).

5.1 Results

ALIVE Business Network

In the ALIVE business network, sources deliver multimedia channels to heterogeneous receivers (e.g., in terms of the capabilities of mobile hosts or the type of network these hosts connect to) through an infrastructure of aggregators. Sources can off-load certain tasks to aggregators, which

increases the sources' scalability and reduces their costs (notably connectivity costs and subscription management).

The ALIVE business network furthermore enables sources to distribute a multimedia channel through multiple aggregators. Some of these aggregators may be bound to specific networks, which allows sources to serve specific regions of the Internet. The distribution of a channel through multiple aggregators also provides flexibility to end-users because they can receive a certain channel from an aggregator of their choice.

The ALIVE business network is a two-tiered network in that it consists of an application-level part (with sources and aggregators) and a network-level part (with providers of Internet access). While this makes the business network more complex, it also provides flexibility advantages because users can independently select aggregators and access providers. The two-tiered approach is furthermore in line with current trends in content distribution.

The aggregators in the ALIVE business network can deliver content channels (e.g., CNN TV) in various configurations. Aggregators typically support a relatively small number of configurations, thus striking a balance between per-user personalization of a channel (e.g., delivering the channel in a configuration that is tailored to the characteristics of a specific user's mobile host) and no personalization at all (i.e., everybody receiving a channel in the same configuration). This approach improves the scalability of aggregators, but might result in users receiving a channel in a suboptimal configuration (e.g., because their network connections provide some extra bandwidth, but not enough to receive the channel in the next higher configuration). A course-grained sampling of the spectrum in which an aggregator can deliver a channel is furthermore in line with current Internet practices in which users can typically receive a multimedia channel at a few bitrates.

To support roaming users, aggregators establish application-level roaming agreements amongst each other. These agreements enable users to receive channels from multiple aggregators (e.g., at different locations) while having a subscription with only a few of them (typically one). Application-level roaming agreements define in which configurations a user can receive channels from a foreign aggregator and can be considered the application-level counter parts of traditional network-level roaming agreements.

ALIVE System

The ALIVE system enables mobile users to roam in an unrestricted manner while continuously receiving a channel. The system transparently switches mobile hosts from one aggregator to another and executes handoffs on the mobile host's network interfaces. The system switches a mobile host to the aggregator that provides a certain channel in the best configuration, where

best is defined by the end-user. This makes the ALIVE system a user-oriented system.

The ALIVE system is scalable because most of its logic resides on mobile hosts. The system's operation is policy-controlled, which enables the stakeholders to flexibly change the rules that the ALIVE system uses to make decisions (e.g., when to look for another available configuration of a channel).

The ALIVE system contains an application-level protocol, which we implemented using the Session Initiation Protocol (SIP) and the Session Description Protocol (SDP), both of which are Internet standards. We deployed our implementation in a testbed with different types of networks, which represents the 'beyond 3G' Internet environment in which the ALIVE protocol is supposed to operate.

Analysis of the ALIVE System

The analysis of our implementation of the ALIVE protocol (i.e., its implementation based on SIP) provides quantitative information on smooth switching, which enables users to seamlessly receive a channel while they roam. Our analysis concentrates on the delay introduced by the ALIVE protocol in a contemporary wireless Internet environment, specifically consisting of 802.11 hotspots and UMTS/GPRS overlays. We focus on the operation of the ALIVE protocol right after a handoff to another 802.11 access provider, which is where the ALIVE protocol typically comes into play. After a handoff, the ALIVE system usually first attempts to discover the configurations in which the user can receive a channel from the local aggregators on the new network. At the edges of 802.11 cells, this may result is a significant delay because of packet loss on the link and the exponential back-off mechanism that SIP uses to recover from such losses.

Our analysis consists of two parts: (1) a heuristic analysis of the application and network-level delay components involved in a typical switch and an estimation of their best-case values, and (2) an empirical analysis of the delay introduced by SIP transactions under various 802.11 network conditions. In this last part, we used an 802.11b network and varied the SNR, the maximum number of 802.11 retransmissions (the retry limit), and the amount of competing traffic (constant bitrate).

In an unloaded network, our experiments show that SIP transactions usually introduce little delay at the edge of an 802.11 cell if the retry limit is at least two. The only exception is a small region (about 2-3 dB wide) at the far end of the cell where the number of SIP transactions that require one or more retransmissions increases rapidly. Our experiments furthermore suggest that a retry limit of more than two does not significantly reduce the delay introduced by SIP transactions in the 2-3 dB region. For a network fully loaded with constant bitrate traffic, the effective retry limits are 4 or 6.

This type of information is also relevant for operators of 802.11 networks because a larger number of 802.11 retransmissions might affect the queuing delays of multimedia packets in the 802.11 MAC.

Our experiments also show that the influence of background traffic alone does not negatively affect the SIP delay with a retry limit of at least two. Finally, we have also experimented with the retransmission behavior of SIP transactions at different transmission rates, which enables us to estimate the edge of the network given a certain transmission rate and SNR. This may be useful information for a rate adaptation scheme that tries to avoid that a SIP-based application needs to run at the very edge of a network.

5.2 Contributions

The contributions of this thesis are:

- A well-defined business network for the distributing live multimedia content in a wireless Internet using multiple intermediaries (aggregators). A business network like this is lacking in systems with objectives similar to the ALIVE system;
- The design, implementation, and validation of a mobile-controlled system that enables mobile hosts to switch to the aggregator that provides a channel in the best way and an implementation of it based on standard Internet protocols. Similar systems only consider switches between proxy servers of the same administrative authority ('intra-aggregator switches') or put most of the system's responsibility in the IP infrastructure;
- An empirical analysis of the delay introduced by SIP transactions over 802.11 links under various network conditions. Until now, this has only been done though simulations; and
- Hints on how to dimension an 802.11 network under varying load conditions and radio qualities.

5.3 Future Work

In this section, we briefly consider future work with respect to the ALIVE business network, the system, and our analysis.

ALIVE Business Network

The ALIVE business network can be detailed and extended in several ways. One possibility to extend the network is to use a hierarchy of aggregators (cf. [Chawathe02]) instead of just one layer of aggregators. Other

possibilities are to consider the impact accounting on the business network (e.g., in application-level roaming agreements) and to study the languages that can be used to describe the agreements in the network (e.g., a right expression language that sources can use to specify in which ways aggregators are allowed to manipulate a channel). Yet another option is to consider the use of a 'universal list of configurations' from which aggregators would pick standardized configurations.

ALIVE System

Future work for the ALIVE system is to detail switches between different types of media servers (e.g., between a SIP server and an RTSP server) and what this would mean for the involved signaling protocols. Another topic could be session mobility, in which users transfer an ongoing multimedia session from one device to another (e.g., from their smart phone to a flat panel television). A further possibility is to investigate the use of media servers that can begin to stream a channel at a specified point in time. This would assist mobile hosts in executing smooth switches, but also requires aggregators to use a delay buffers in case an inbound mobile host wants to begin receiving a channel some time 'in the past'. This topic would also require the ALIVE system to be able to discover if media servers support this capability. Two final topics are the implementation of the configuration notification service (e.g., using SIP eventing [Roach02]) and the design of the ALIVE system based on a completely different principle (e.g., thin client).

ALIVE Analysis

The analysis of the ALIVE protocol could be extended in several dimensions. One possibility is to consider the performance of the system using different configuration discovery policies (e.g., proactive versus reactive) and under different circumstances (e.g., velocity and battery drain). Another possibility is to conduct additional experiments, for instance in a less controlled environment, using variable bitrate background traffic, using multiple hosts at different transmission rates, another interval between consecutive SIP transactions, and so on.

Samenvatting

In dit proefschrift behandelen we het efficiënt distribueren van live en aangekondigde multimedia stromen (bijvoorbeeld radio- en televisie-uitzendingen) naar mobiele gebruikers via een grenzeloze Internet omgeving. Het doel van dit proefschrift is het ontwerpen en ontwikkelen van een content delivery systeem dat (1) eigenaars van live multimedia informatie in staat stelt hun informatie af te leveren bij een groot aantal heterogene ontvangers en (2) ontvangers in staat stelt bepaalde informatie continu te blijven ontvangen, onafhankelijk van hun locatie of het netwerk dat ze gebruiken.

Eerdere studies op het gebied van het efficiënt distribueren van multimedia stromen via het Internet laten zien dat het mogelijk is deze stromen te verspreiden via een overlay netwerk dat bestaat uit meerdere gedistribueerde proxy servers. In dit proefschrift breiden we dit concept uit naar het verspreiden van live en aangekondigde multimedia informatie via meerdere *aggregators*. Een aggregator is een intermediair die multimedia informatie verzamelt van verschillende zenders en deze informatie vervolgens aanbiedt aan mobiele gebruikers. Een aggregator maakt daarbij gebruik van een verzameling media servers.

Doordat dezelfde informatie via meerder aggregators beschikbaar is kunnen mobiele gebruikers van de ene naar de andere aggregator schakelen. Hierdoor ontvangen mobiele gebruikers dezelfde informatie afwisselend van verschillende aggregators.

Het dekkingsgebied van een aggregator kan beperkt zijn tot een bepaald aantal netwerken, wat betekent dat het omschakelen naar een dergelijke aggregator vereist dat de mobiele gebruiker ook overgaat naar een netwerk dat onderdeel is van het dekkingsgebied van de aggregator.

Het systeem dat de apparatuur van mobiele gebruikers omschakelt naar een andere aggregator is het ALIVE systeem, wat staat voor **A**ggregator Switching System for Mobile Receivers of **Live** Multimedia Content. We richten ons vooral op de 'front-end' van het ALIVE systeem, wat bestaat uit

mobiele gebruikers van mobiele apparaten, aggregators en draadloze netwerken. In het bijzonder richten we ons op de signaleringsinteracties tussen mobiele apparaten en aggregators. De details van multimedia informatie zelf zijn geen onderdeel van ons werk.

Het ontwerp van het ALIVE systeem is gebaseerd op het *ALIVE business netwerk*. Dit is een netwerk van business rollen, dat bijvoorbeeld bestaat uit rollen zoals 'aggregator' en 'eindgebruiker'. Het netwerk beschrijft de relaties die kunnen bestaan tussen de domeinen die betrokken zijn bij het verspreiden en ontvangen van live multimedia informatie via meerdere aggregators. Het ALIVE business netwerk bestaat uit een applicatie-level deel (een overlay die bestaat uit zenders en aggregators van multimedia informatie) en een netwerk-level deel (bestaand uit aanbieders van basis Internet toegang). Deze opdeling komt overeen met huidige trends op het gebied van content distributie. We definiëren de eigenschappen van de relaties in het ALIVE business netwerk in de vorm van overeenkomsten.

Het ALIVE business netwerk gebruikt de notie van een van een kanaal voor een bepaald stuk multimedia informatie (bijvoorbeeld een live uitzending van het NOS journaal). De aggregators in het ALIVE business netwerk kunnen een kanaal in verschillende configuraties aanbieden om zo het aantal ontvangers van het kanaal verder te vergroten. Een configuratie levert een kanaal op een bepaald perceptueel kwaliteitsniveau en vereist een goed gedefinieerde hoeveelheid resources (bijvoorbeeld netwerkbandbreedte). Aggregators kunnen er voor kiezen een relatief klein aantal configuraties te ondersteunen om zo een balans te creëren tussen personalisatie voor individuele gebruikers (bijvoorbeeld door middel van configuraties die zijn afgestemd op de huidige bandbreedte beschikbaar voor een specifiek mobiel apparaat) en helemaal geen personalisatie (iedereen ontvangt een kanaal in één en dezelfde configuratie).

Aggregators zetten onderling zogenaamde applicatie-level roaming overeenkomsten op om mobiele gebruikers te ondersteunen. Deze overeenkomsten stellen gebruikers in staat kanalen te ontvangen van meerdere aggregators (bijvoorbeeld op verschillende locaties) terwijl ze een abonnement hebben met slechts enkele aggregators (typisch één). Applicatie-level roaming overeenkomsten definiëren in welk configuraties gebruikers een kanaal kunnen ontvangen van aggregators waarbij ze geen abonnement hebben (foreign aggregators).

Een aggregator kan gebonden zijn aan een bepaalde verzameling netwerken door middel van binding-overeenkomsten met leveranciers van Internet toegang. In het ALIVE business netwerk heet een aggregator die betrokken is in een binding-overeenkomst een locale aggregator. De reden hiervoor is dat de binding-overeenkomst de beschikbaarheid van de aggregator beperkt tot de netwerken van de betrokken Internet leverancier. Een leverancier van Internet toegang kan locale aggregators gebruiken om

exclusieve kanalen of configuraties van kanalen aan te bieden aan gebruikers die via de één van zijn netwerken een verbinding met het Internet maken (vergelijk de walled-garden modellen die cellulaire operators tegenwoordig typisch hanteren).

Het *ALIVE systeem* stelt mobiele gebruikers in staat om zich op een onbeperkte manier te verplaatsen terwijl zijn een kanaal ontvangen. Het systeem schakelt mobiele apparaten automatisch van de ene aggregator naar de andere en verbindt ze met een netwerk waarin de nieuwe aggregator beschikbaar is. Dit alles vindt grotendeels transparant voor de eindgebruiker plaats. Het systeem schakelt een mobiel apparaat naar een aggregator die een bepaald kanaal in de beste configuratie aanbiedt, waarbij 'de beste' wordt bepaald door de voorkeuren van de gebruiker. Dit maakt het ALIVE systeem een gebruikersgeorienteerd systeem.

Het ALIVE systeem is schaalbaar omdat de meeste intelligentie op mobiele apparaten zit (mobiel-gecontrolleerd schakelen). De werking van het systeem is daarnaast gebaseerd op policies. Dit stelt de stakeholders in het ALIVE business netwerk in staat om op een flexibele manier de regels te veranderen die het ALIVE systeem gebruikt om schakelbeslissingen te maken (bijvoorbeeld wanneer het systeem op zoek gaat naar een alternatieve configuratie voor een bepaald kanaal).

Het ALIVE systeem bevat een applicatieprotocol dat we hebben gerealiseerd met behulp van het Session Initiation Protocol (SIP) en het Session Description Protocol (SDP). Beide protocollen zijn Internet standaarden. We hebben onze implementatie ingezet in een kleinschalige testomgeving met meerdere typen netwerken. Deze netwerken representeren de 'beyond 3G' Internetomgeving waarin het ALIVE systeem zou moeten draaien.

Middels een *analyse* van onze SIP implementatie van het ALIVE protocol hebben we kwantitatieve informatie verkregen over hoe schakelingen tussen aggregators soepel kunnen worden uitgevoerd. Onze analyse richt zich op de extra vertraging die het ALIVE protocol introduceert in een hedendaagse draadloze Internetomgeving, in het bijzonder een omgeving die bestaat uit 802.11 hotspots en UMTS/GPRS overlays. We richten ons verder op de werking van het ALIVE protocol meteen na een wisseling naar een andere 802.11 aanbieder omdat dit typisch het moment is waar het ALIVE protocol in werking treedt. Na een wisseling van netwerkaanbieder probeert het ALIVE systeem namelijk typisch eerst te ontdekken in welke configuraties de gebruiker een kanaal kan ontvangen van de locale aggregators op het nieuwe netwerk. Aan de randen van 802.11 cellen kan dit leiden tot significante vertragingen vanwege het exponentiële back-off mechanisme dat SIP gebruikt om te herstellen van pakketverlies.

Onze analyse bestaat uit twee delen: (1) een heuristische analyse van de applicatie- en netwerk-level vertragingscomponenten van een typische

aggregatoromschakeling en een schatting van hun waarden, en (2) een empirische analyse van de vertraging die SIP transacties introduceren onder verschillende 802.11 netwerkcondities.

Op basis van onze implementatie en onze metingen concluderen we dat het ALIVE systeem een realiseerbaar systeem is dat een duidelijke bijdrage levert een de visie van ‘multimedia-anywhere’.

References

[3GPP99] 3GPP, “Automatic Establishment of Roaming Relationships”, Technical Report 22.971, version 3.1.1, April 1999

[Abilene04] Abilene one-way delay statistics, http://abilene.internet2.edu/ami/owamp_status.cgi/

[Aguayo04] D. Aguayo, J. Bicket, S. Biswas, G. Judd, R. Morris, “Link-level Measurements from an 802.11b Mesh Network”, SIGCOMM’04, Portland, USA, Aug-Sept 2004

[Amir95] E. Amir, S. McCanne, and H. Zhang, “An Application Level Video Gateway”, Proc. of ACM Multimedia, San Fransisco, USA, Nov. 1995

[Amir98] E. Amir, S. McCanne, R. Katz, “An Active Service Framework and its Application to Real-time Multimedia Transcoding”, ACM SIGCOMM’98, Vancouver, Canada, Sept. 1998

[Balachandran97] A. Balachandran, A. Campbell, M. Kounavis, “Active Filters: Delivering Scaled Media to Mobile Devices”, 7th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'97), St. Louis, USA, May 1997

[Banerjee03] N. Banerjee, S. K. Das, “Hand-off Delay Analysis in SIP-based Mobility Management in Wireless Networks”, Proceedings of IEEE International Workshop on Wireless, Mobile Ad hoc Networks (WMAN'03), Nice, France, Apr 2003

[Banerjee04] N. Banerjee, W. Wu, K. Basu, S. K. Das, “Analysis of SIP-Based Mobility Management in 4G Wireless Networks”, Computer Communications, Vol 27, No. 8, pp 697-707, 2004

[Brewer98] E. Brewer, R. Katz, Y. Chawathe, S. Gribble, T. Hodes, G. Nguyen, M. Stemm, T. Henderson, E. Amir, H. Balakrishnan, A. Fox, V. Padmanabhan, S. Seshan, “A Network Architecture for Heterogeneous Mobile Computing”, IEEE Personal Communications, Oct. 1998

[Cain03] B. Cain, A. Barbir, R. Nair, O. Spatscheck, “Known CN Request-Routing Mechanisms”, Internet Draft, draft-ietf-cdi-known-request-routing-03.txt, April 2003

[Calhoun03] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, “Diameter Base Protocol”, Internet Draft, April 2003, draft-ietf-aaa-diameter-17.txt

[Camarillo03] G. Camarillo, R. Kantola, H. Schulzrinne, “Evaluation of Transport Protocols for the Session Initiation Protocol”, IEEE Network, Sept./Oct. 2003, pp. 40-46

[Campbell00] A. Campbell, J. Gomez, S. Kim, A. Valkó, C. Wan, Z. Turányi, "Design, Implementation, and Evualation of Cellular IP", IEEE Personal Communications, August 2000

[Chawathe02] Y. Chawathe, "Scattercast: An Adaptable Broadcast Distribution Framework", Special Issue of ACM Multimedia Distribution, 2002

[Chennikara02] J. Chennikara, W. Chen, A. Dutta, O. Altintas, "Application-Layer Multicast for Mobile Users in Diverse Networks", Proceedings IEEE Globecom 2002, Taipei, Taiwan, November 2002

[Cheung96] S. Cheung, M. Ammar, X. Li, "On the use of destination set grouping to improve fairness in multicast video distribution", proceedings IEEE Infocom '96, San Francisco, USA, March 1996, pp. 553-560

[Clark88] D. Clark, "The Design Philosophy of the DARPA Internet Protocols", ACM SIGCOMM, Sept. 1988

[Cox99] M. Cox, R. Davidson, "Concepts, Activities and Issues of Policy-based Communications Management", BT Technology Journal, Vol. 17, Issue 3, July 1999, pp. 155-169

[Curcio 01] I. Curcio, M. Lundan, "Study of Call Setup in SIP-Based Videotelephony", 5th World Multi-Conference on Systemics, Cybernetics, and Informatics (SCI 2001), Orlando, Florida, USA, July 2001

[Curcio 02] I. Curcio, M. Lundan, "SIP Call Setup Delay in 3G Networks", 7th International Symposium on Computers and Communications (ISCC'02), Taormina-Giardini Naxos, Italy, July 2002

[Day01] M. Day, B. Cain, G. Tomlinson, P. Rzewski, "A Model for Content Interworking (CDI)", Internet Draft, draft-day-cdnp-model-07.txt, Oct. 2001

[DeCleyn04] P. De Cleyn, N. Van den Wijngaert, L. Cerdá, C. Blondia, "A smooth handoff scheme using IEEE802.11 triggers – design and implementation", International Journal of Computer and Telecommunications Networking, Computer Networks and ISDN (Elsevier), Volume 45, Issue 3, pp 345-361, June 2004

[DOLMEN98] DOLMEN Consortium, "Open Service Architecture for Mobile and fixed environments (OSAM)", Final Release, Version 4.0, July 1998

[Donovan02] S. Donovan, J. Rosenberg, "Session Timers in the Session Initiation Protocol (SIP)", Internet Draft, Nov. 2002, draft-ietf-sip-session-timer-10.txt

[Doufexi03] A. Doufexi, E. Tameh, A. Nix, S. Armour, A. Molina, "Hotspot Wireless LANs to Enhance the Performance of 3G and Beyond Cellular Networks", IEEE Communications Magazine, July 2003, pp. 58-65

[Drew01] N. Drew, M. Dillinger, "Evolution Toward Reconfigurable User Equipment", IEEE Communications Magazine, Feb. 2001

[Droms99] R. Droms, "Automated Configuration of TCP/IP with DHCP", IEEE Internet Computing, July-August 1999

[Dutta02] A. Dutta, H. Schulzrinne, S. Das, A. McAuley, W. Chen, O. Altintas, "MarconiNet supporting Streaming Media over Localized Wireless Multicast", M-Commerce 2002 Workshop, Atlanta, USA, September 2002

[Eckhardt96] D. Eckhardt, P. Steenkiste, "Measurement and Analysis of the Error Characteristics of an In-building Wireless Network", Computer Communications Review, October 1996, pp. 243-254

[expect] http://expect.nist.gov/

[Eyers00] T. Eyers, H. Schulzrinne, "Predicting Internet Telephony Call Setup Delay," Proc. 1st IP Telephony Wksp., Berlin, Germany, Apr. 2000

[Floyd02] S. Floyd, L. Daigle, "IAB Architectural and Policy Considerations for Open Pluggable Edge Services", RFC 3238, January 2002

[FreeRadius] Free Radius webpage, http://www.freeradius.org/

[Gao03] L. Gao, Z.-L. Zhang, D. Towsley, "Proxy-Assisted Techniques for Delivering Continuous Multimedia Streams", IEEE/ACM Transactions on Networking, December 2003, pp. 884-894

[Gast02] M. Gast, "802.11 Wireless Networks – The Definitive Guide", O'Reilly, 2002

[Guttman99] E. Guttman, C. Perkins, J. Veizades, M. Day, "Service Location Protocol, Version 2", RFC 2608, June 1999

[Haardt00] M. Haardt, W. Mohr, "The Complete Solution for Third-Generation Wireless Communications: Two Modes on Air, One Winning Strategy", IEEE Personal Communications., December 2000

[Halteren99] A. van Halteren, C. Hesselman, G. Koprinkov, G. Túquerres, D. de Vries, I. Widya, "QoS Architecture – Amidst Perspective", Amidst Deliverable 3.1.2, November 1999, http://amidst.ctit.utwente.nl/workpackages/wp3/documents/d312.pdf

[Handley98] M. Handley, V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998

[Haratcherev04] I. Haratcherev, K. Langendoen, R. Lagendijk, H. Sips, "Hybrid Rate Control for IEEE 802.11", ACM International Workshop on Mobility Management and Wireless Access Protocols (MobiWac), Philadelphia, PA, October 2004

[Helmy00] A. Helmy, "A Multicast-based Protocol for IP Mobility Support", ACM Second International Workshop on Networked Group Communication (NGC 2000), Palo Alto, USA, November 2000

[Hesselman01] C. Hesselman, H. Eertink, A. Peddemors, "Multimedia QoS Adaptation for Inter-tech Roaming", Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC'01), Hammamet, Tunisia, July 2001

[Hesselman02] C. Hesselman, I. Widya, H. Eertink, E. Huizer, "A Comprehensive Framework for Broadcasting Multimedia Content in the Future Mobile Internet", Proceedings of the 2nd IEEE Workshop on Applications and Services in Wireless Networks (ASWN'02), Paris, France, July 2002

[Hesselman03] C. Hesselman, H. Eertink, I. Widya, E. Huizer, "A Mobility-aware Broadcasting Infrastructure for a Wireless Internet with Hotspots", Proceedings of the First ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH'03), San Diego, USA, September 2003

[Hesselman05] C. Hesselman, H. Eertink, I. Widya, E. Huizer, "Delivering Live Multimedia Streams to Mobile Hosts in a Wireless Internet with Multiple Content Aggregators", to appear in Mobile Networks and Applications journal (Kluwer Wireless), Special Issue on Wireless Mobile Applications and Services on WLAN Hotspots, Summer 2005

[Hoene03] C. Hoene, A. Günther, A. Wolisz, "Measuring the Impact of Slow User Motion on Packet Loss and Delay over IEEE 802.11b Wireless Links", In Proc. of Workshop on Wireless Local Networks (WLN) 2003, Bonn, Germany, Oct. 2003

[Hsieh03] H-Y. Hsieh, K-H. Kim, Y. Zhu, R. Sivakumar, "A Receiver-Centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces", Proc. MobiCom 2003, San Diego, USA, September 2003

[jtg] jtg v1.69, http://www.cs.helsinki.fi/u/jmanner

[Kamilova05] M. Kamilova, C. Hesselman, I. Widya, E. Huizer, "Adding Policy-based Control to Mobile Hosts Switching between Streaming Proxies" (short paper), Proc. of the Sixth IEEE Intl. Workshop on Policies for Distributed Systems and Networks (POLICY 2005), Stockholm, Sweden, June 2005.

[Karrer01] R. Karrer, T. Gross, "Dynamic Handoff of Multimedia Streams", In Proc. NOSSDAV '01, pages 125-133, Danfords on the Sound, Port Jefferson, New York, June 2001.

[Kempf00] J. Kempf, J. Goldschmidt, "Mobile Code for Network Service Access", INET 2000, Yokohama, Japan, July 2000, http://www.isoc.org/inet2000/cdproceedings/3c/3c_2.htm

[Kim01] K. Kim, H. Lee, K. Chung, "A Distributed Proxy Server System for Wireless Mobile Web Services", 15th International Conference on Information Networking (ICOIN'01), Beppu City, Oita, Japan, January-February 2001

[kismet] http://www.kismetwireless.net/

[Kleinrock03] L. Kleinrock, "An Internet Vision: the Invisible Global Instrastructure", AdHoc Networks Journal, Vol. 1, No. 1, pp. 3-11, July 2003

[Køien03] G. Køien, T. Haslestad, "Security Aspects of 3G-WLAN Interworking", IEEE Communications Magazine, November 2003, pp. 82-88

[Kouvelas98] I. Kouvelas, V. Hardman, J. Crowcroft, "Network Adaptive Continuous-Media Applications Through Self Organised Transcoding", Proc. Network and Operating Syst. Support for Digital Audio and Video (NOSSDAV'98), July 1998, Cambridge, UK

[Kueh04] V. Kueh, R. Tafazolli, B. Evans, "Performance of VoIP Call Set-up Over Satellite-UMTS Using Session Initiation Protocol", European Wireless Conference 2004, Barcelona, Spain, February 2004

[Kutscher03] Kutscher, Ott, Bormann, "Session Description and Capability Negotiation", Internet Draft, draft-ietf-mmusic-sdpng-06.txt, March 2003

[Kwon02] T. Kwon, M. Gerla, S. Das, S. Das, "Mobility Management for VoIP Service: Mobile IP vs. SIP", IEEE Wireless Communications, October 2002, pp. 2-11

[Landfeldt99] B. Landfeldt, T. Larsson, Y. Ismailov, A. Seneviratne, "SLM, a Framework for Session Layer Mobility Management", In Proc. IEEE ICCCN, October 1999

[Latvakoski02] E. Latvakoski, P. Laurila, "Application-based Access System Selection Concept for All-IP Mobile Terminals", IEEE Globecom 2002, Nov 2002, Taipei, Taiwan

[Li02] M. Li, M. Claypool, M. Rinicki, "MediaPlayer™ versus RealPlayer™ – A Comparison of Network Turbulence", Proceedings of the ACM SIGCOMM Internet Measurements Workshop, Marseille, France, November 2002

[Liao99] W. Liao, "Mobile Internet Telephony Protocol (MITP): an Application-Layer Protocol for Mobile Internet Telephony Services", Proc. IEEE ICC'99, Vancouver, Canada, June 1999

[Maltz98] D. Maltz, P. Bhagwat, "MSOCK: An Architecture for Transport Layer Mobility", In Proc. IEEE Infocom, March 1998

[Markoulidakis97] G. Markoulidakis, G. Lyberopoulos, D. Tsirkas, E. Sykas, "Inter-Operator Roaming Scenarios for Third Generation Mobile Telecommunication Systems", 2nd IEEE Symposium on Computers and Communications (ISCC'97), Alexandria, Egypt, July 1997

[Marshall03] W. Marshall, "Private Session Initiation Protocol (SIP) Extensions for Media Authorization", RFC 3313, Jan. 2003

[McCanne96] S. McCanne, V. Jacobson, M. Vetterli, "Receiver-driven Layered Multicast", Proc. of ACM SIGCOMM, Stanford, USA, August 1996

[Microsoft00] Microsoft Corporation, "Universal Plug and Play Device Architectute", Version 1.0, June 2000, http:// www.upnp.org/download/UPnPDA10_20000613.htm

[Mishra03] A. Mishra, M. Shin, W. Arbaugh, "An Empirical Analysis of the IEEE 802.11 MAC Layer Handoff Process" ACM SIGCOMM Computer Communication Review, Volume 33, Number 2, April 2003, pp. 93-102

[Mishra04] A. Mishra, M. Shin, N. Petroni, T. Clancy, W. Arbaugh, "Proactive Key Distibution Using Neighbor Graphs", IEEE Wireless Communications, Feb. 2004, pp. 26-36

[Moessner02] K. Moessner, S. Hope, P. Cook, W. Tuttlebee, R. Tafazolli, "The RMA - A Framework for Reconfiguration of SDR Equipment", IEICE Transactions on Communication, Vol. E85-B, No.12, pp.2573-2580, December 2002

[Nakajima03] N. Nakajima, A. Dutta, S. Das, H. Schulzrinne, "Handoff Delay Analysis for SIP Mobility in IPv6 Testbed", IEEE International Conference on Communications, Anchorage, Alaska, USA, May 2003

[Nomura03] Y. Nomura, R. Walsh, J. Ott, H. Schulzrinne, "Protocol Requirements for Internet Media Guides", Internet Draft, draft-ietf-mmusic-img-req-00.txt, September 2003

[Norton02a] W. Norton, "Internet Service Providers and Peering", White Paper (v2.5), February 2002, http://www.equinix.com/pdf/whitepapers/PeeringWP.2.pdf

[Norton02b] W. Norton, "A Business Case for ISP Peering", White Paper (v1.3), February 2002, http://www.equinix.com/pdf/whitepapers/Business_case.pdf

[OpenSIP] oSIP webpage, http://www.gnu.org/software/osip/

[Pack02] S. Pack, Y. Choi, "Fast Inter-AP Handoff Using Predictive Authentication Scheme in a Public Wireless LAN", Networks 2002 (Joint ICN 2002 and ICWLHN 2002), Atlanta, USA, pp.15-26, August 2002

[Pahlavan00] K. Pahlavan, P. Krishnamurthy, A. Hatami, M. Ylianttila, J. Makela, R. Pichna, J. Vallström, "Handoff in Hybrid Mobile Data Networks", IEEE Personal Communications, April 2000

[Peddemors04] A. Peddemors, H. Zandbelt, M. Bargh, "A Mechanism for Host Mobility Management Supporting Application Awareness", MobiSys'04, Boston, Massachusetts, USA, June 2004, pp. 231-244

[Peterson03] J. Peterson, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", Internet Draft, Feb. 2003, draft-ietf-sip-identity-01.txt

[Plagemann03] T. Plagemann, V. Goebel, L. Mathy, N. Race, M. Zink, "Towards Scalable and Affordable Content Distribution Services", Proc. 7th International Conference on Telecommunications (ConTEL 2003), Zagreb, Croatia, June 2003

[Pollini96] G. Pollini, "Trends in Handover Design", IEEE Communications Magazine, March 1996, also available through IEEE Communications Surveys, http://www.comsoc.org/pubs/surveys/pollini/pollini-org.html

[Punnoose01] R. Punnoose, R. Tseng, D. Stancil, "Experimental Results for Interference between Bluetooth and IEEE 802.11b DSSS Systems", IEEE Vehicular Society Conference, October 2001

[Ramjee99] R. Ramjee, T. La Porta, S. Thuel, K. Varadhan, "HAWAII: A Domain-based Approach for Supporting Mobility in Wide-area Wireless Networks", Seventh International Conference on Network Protocols, Toronto, Canada, 1999

[Rasmusson04] J. Rasmusson, F. Dahlgren, H. Gustafsson, T. Nilsson, "Multimedia in mobile phones - The ongoing revolution", Ericsson Review no. 02, 2004, http://www.ericsson.com/mobilityworld/sub/articles/other_articles/05jan05

[Roach02] A. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002

[Rosenberg02a] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002

[Rosenberg02b] J. Rosenberg, H. Schulzrinne, "An Offer/Answer Model with the Session Description Protocol (SDP)", RFC 3264, June 2002

[Rosenberg98] J. Rosenberg, H. Schulzrinne, "Internet Telephony Gateway Location", Proc. IEEE Infocom'98, March 1998

[Roy02] S. Roy, B. Shen, V. Sundaram, R. Kumar, "Application Level Hand-off Support for Mobile Media Transcoding Sessions", NOSSDAV'02, Miami Beach, Florida, May 2002

[Saltzer84] J. Saltzer, D. Reed, D. Clark, "End-to-end Arguments in System Design", ACM Transactions on Computer Systems, Vol. 2, Issue 4, Nov. 1984, pp. 277-288

[Satyanarayanan01] M. Satyanarayanan, "Pervasive computing: vision and challenges", IEEE Personal Communications, Aug 2001, pp. 10-17

[Schulzrinne02] H. Schulzrinne, "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", RFC 3361, August 2002

[Schulzrinne96a] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996

[Schulzrinne96b] H. Schulzrinne, "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, January 1996

[Schulzrinne98] H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998

[SDR] SDR at UCL, http://www-mice.cs.ucl.ac.uk/multimedia/software/sdr/

[Seneviratne98] A. Seneviratne, B. Sarikaya, "Cellular Networks and Mobile Internet", Computer Communications, Sept. 1998

[Shin04] S. Shin, A. Rawat, H. Schulzrinne, "Reducing MAC Layer Handoff Latency in IEEE 802.11 Wireless LANs", MobiWac'04, Philadelphia, Pennsylvania, USA, Oct. 2004

[Snoeren00] A. Snoeren, H. Balakrishnan, “An end-to-end Approach to Host Mobility”, ACM MobiCom, August 2000

[Snoeren01] A. Snoeren, H. Balakrishnan, F. Kaashoek, “Reconsidering Internet Mobility”, In Proc. 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), Elmau/Oberbayern, Germany, May 2001

[Snoeren01b] A. Snoeren, D. Andersen, H. Balakrishnan, “Fine-Grained Failover Using Connection Migration”, Proc. USENIX USITS, San Fransisco, USA, March 2001

[Solomon98] J. Solomon, “Mobile IP — The Internet Unplugged”, Prentice Hall, 1998

[Stemm98] M. Stemm, R. Katz, “Vertical Handoffs in Wireless Overlay Networks”, ACM Mobile Networking, Special Issue on Mobile Networking and Internet, Spring 1998

[Sultan02] F. Sultan, K. Srinivasan, D. Iyer, L. Iftode, “Migratory TCP: Connection Migration for Service Continuity in the Internet”, Proc. IEEE ICDCS, Vienna, Austria, July 2002

[Tan99] C. Tan, S. Pink, K. Lye, “A Fast Handoff Scheme for Wireless Networks”, Proc. 2nd ACM Workshop on Wireless Mobile Multimedia (WoWMoM’99), Seattle, Aug. 1999

[tcpdump] http://www.tcpdump.org/

[TINA97] TINA Consortium, “Business network and Reference Points”, Chapter 2, May 1997, http://www.tinac.com/specifications/documents/bm_rp.pdf

[Tripathi98] N. Tripathi, J. Reed, H. VanLandingham, “Handoff in Cellular Systems”, IEEE Personal Communications, Dec. 1998

[Trossen03] D. Trossen, H. Chaskar, “Seamless Mobile Applications across Heterogeneous Internet Access”, IEEE ICC 2003, May 2003, Anchorage, Alaska, USA

[Vatn03] J.-O. Vatn, “An Experimental Study of IEEE 802.11b Handover Performance and its Effect on Voice Traffic”, Technical Report, Royal Institute of Technology, Stockholm, Sweden, July 2003

[Vatn98] J.-O. Vatn, G. Maguire, “The effect of using co-located care-of addresses on macro handover latency”, Fourteenth Nordic Tele-traffic Seminar (NTS 14), Lyngby, Denmark, August 1998

[Velayos03] H. Velayos, G. Karlsson, “Techniques to Reduce IEEE 802.11b MAC Layer Handover Time” Technical Report, Royal Institute of Technology, Stockholm, Sweden, April 2003

[Verhoosel03] J. Verhoosel, R. Stap, A. Salden, "A Generic Business network for WLAN Hotspots – A Roaming Business Case in The Netherlands", Proceedings of the first ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH'03), San Diego, USA, September 2003

[Vernick01] M. Vernick, S. Bryden, M. Condry, G. Disher, J. Straley, W. Walkoe, “Requirements for End-To-End Delivery of Broadband Content”, Broadband Content Delivery Forum, Oct. 2001

[VIC] VIC at UCL, http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/

[Wang99] H. Wang, R. Katz, J. Giese, “Policy-Enabled Handoffs Across Heterogeneous Wireless Networks”, 2nd IEEE Workshop on Mobile Computing and Applications (WMCSA 1999), New Orleans, USA, February 1999

[Wang04] X. Wang, "MPEG-21 Rights Expression Language: enabling interoperable digital rights management", IEEE Multimedia, Vol. 11, Issue 4, Oct.-Dec. 2004

[Wedlund99] E. Wedlund, H. Schulzrinne, "Mobility Support Using SIP", 2nd ACM/IEEE Int. Conf. on Wireless and Mobile Multimedia (WoWMoM'99), Seattle, USA, Aug. 1999

[Wee03] S. Wee, J. Apostolopoulos, W.-T. Tan, S. Roy, "Research and Design of a Mobile Streaming Media Content Delivery Network", International Conference on Multimedia and Expo (ICME'03), Baltimore, Maryland, USA, July 2003

[Westerinen01] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser "Terminology for Policy-Based Management", RFC 3198. November 2001

[Wired04] "Let the Web Games Begin", Wired Magazine, August 2004, http://www.wired.com/news/culture/0,1284,64562,00.html

[Wong03] K. Wong, V. Varma, "Supporting Real-Time IP Multimedia Services in UMTS", IEEE Communications Magazine, November 2003

[Wu97] L. Wu, R. Sharma, B. Smith, "Thin Streams: An Architecture for Multicast Layered Video", 7th Intl. Workshop on Network an Operating Systems Support for Digital Audio and Video (NOSSDAV97), St. Louis, USA, May 1997

[Xiao99] X. Xiao, L. Ni, "Internet QoS: A Big Picture", IEEE Network, March/April 1999, pp. 8-18

[MobileIT04] Mobile IT Forum, "Flying Carpet 2.0: report on the activities of the 4th Generation Mobile Communications Committee", Tokio, April 2004, http://www.mitf.org/public_e/archives/Flying_Carpet_Ver200.pdf

[Xu00] D. Xu, K. Nahrstedt, "Supporting Multimedia Service Polymorphism in Dynamic and Heterogeneous Environments", Technical Report UIUCDCS-R-2000-2159, University of Illinois at Urbana-Champaign, USA, October 2000

[Yeadon96] N. Yeadon, F. Garcia, D. Hutshison, D. Shepherd, "Filters: QoS Support Mechanisms for Multipeer Communications", IEEE Journal on Selected Areas in Comm., Sept. 1996

[Zenel97] B. Zenel and D. Duchamp, "A General Purpose Proxy Filtering Mechanism Applied to the Mobile Environment", Proc. 3rd ACM/IEEE Intl. Conference on Mobile Computing and Networking, Budapest, Hungary, Sept. 1997

[Zhuang03] W. Zhuang, Y.S. Gan, K.J. Loh, K.C.Chua, "Policy-based QoS Management Architecture in an Integrated UMTS and WLAN Environment", IEEE Communications Magazine, November 2003, pp. 118-12