

# **Design and Implementation of a High Availability SIP Server Architecture**

Diplomarbeit

Nils Ohlmeier

Berlin, 9 Juli 2003

Technische Universität Berlin

Fachbereich IV

Institut für Telekommunikationssysteme

Fachgebiet Offene Kommunikationssysteme (OKS)

Franklinstrasse 28-29

D-10587 Berlin

Betreuer: Dr. habil Thomas Magedanz

Assistierender Betreuer: Prof. Dr. Dr. h.c. Radu Popescu-Zeletin

Matrikel Nummer: 18 19 46



# **Design and Implementation of a High Availability SIP Server Architecture**

Diplomarbeit

Nils Ohlmeier

Berlin, 9 July 2003



# **Eidesstattliche Erklärung**

Hiermit erklärt der Autor an Eides statt, dass er die Arbeit ohne unerlaubte Hilfsmittel und unter ausschließlicher Verwendung der genannten Quellen angefertigt hat.

Berlin, den 9 Juli 2003

Nils Ohlmeier



# Zusammenfassung

Die Verfügbarkeit von SIP-basierten Diensten im Internet erreicht heutzutage nur 98 Prozent, was deutlich unter den berühmten fünf Neunen des PSTN liegt. Wir schlagen daher, um die Gesamtverfügbarkeit zu verbessern, den Einsatz von geographisch verteilten Föderationen von Servern vor. Wir analysieren die existierenden Hochverfügbarkeitstechniken und die Nutzung von Anycast für SIP. Wir empfehlen die Anwendung unserer SIP-basierten Replikation, um die Server zu synchronisieren, welches die Erreichbarkeit der Nutzer deutlich erhöht. Eine Auswahl von Lösungen ermöglicht des weiteren die Authentifizierung der Benutzer, ohne deren Passwort zu wissen. Die vorhandene Implementierung für den SIP Express Router ermöglicht den realen Einsatz von Föderationen.

**Schlüsselwörter:** SIP, Voice over IP, Verfügbarkeit, verteilt, Replikation, Authentifizierung, Anycast



# Abstract

The availability of SIP-based services on the Internet reaches only 98 percent, significantly below the five nines from PSTN. We propose to use geographically distributed federations of servers to improve the overall availability. We analyse the existing HA techniques and the use of Anycast for SIP. We propose to use our SIP-based replication to keep the servers synchronized, which improves the reachability of the users. Furthermore, a range of solutions enables the authentication of the users without knowing their passwords. The implementation, available for the SIP Express Router, enables the deployment of federations.

**Keywords:** SIP, Voice over IP, availability, distributed, replication, authentication, Anycast



# Acknowledgments

First of all, I would like to thank my advisor Dipl.-Ing. Jiri Kuthan for his support before and during the creation of this thesis, a lot of inspiring discussions and some constructive criticism. He deserves a large part of the laurels for this work, for his continuous patience and guidance.

He and Dr.-Ing. Dorgham Sisalem, the head of the Mobis Competence Center at Fraunhofer Gesellschaft Fokus, have earned my thanks for letting me work in a very inspiring and challenging environment with a lot of freedom. Further, I would like to thank the iptel.org team, especially Jan Janak, for suggestive discussions and nice teamwork.

Furthermore, I have to thank the three other members of our "studies quartet", which eased and accelerated my studies a lot and made my studies most of the times a little bit more enjoyable.

My gratitude goes finally to my family, for always supporting me during my studies and work and giving me all the freedom I desired.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Objectives . . . . .	3
1.3	Scope . . . . .	5
1.4	Document Structure . . . . .	5
<b>2</b>	<b>State of the Art</b>	<b>7</b>
2.1	PSTN . . . . .	7
2.2	Internet . . . . .	7
2.2.1	Redundancy and Cluster . . . . .	7
2.2.2	IP as Single Point of Failure . . . . .	8
2.2.3	DNS . . . . .	8
2.2.4	Multicast . . . . .	8
2.2.5	Anycast . . . . .	8
2.2.6	rserpool . . . . .	9
2.3	SIP / VoIP . . . . .	9
2.3.1	SIP Availability Today . . . . .	9
2.3.2	Intelligence in the Network . . . . .	10
2.3.3	Other SIP Products . . . . .	10
2.3.4	H.323 . . . . .	11
2.3.5	SIP Digest Authentication . . . . .	11
<b>3</b>	<b>Requirements for HA SIP services</b>	<b>14</b>
3.1	SIP Dependences . . . . .	14
3.2	Error Types . . . . .	15
3.2.1	Software, Hardware and Required Services . . . . .	15

3.2.2	Environment	15
3.2.3	IP-Connectivity	16
3.2.4	Malicious Attacks	16
3.2.5	Planned Downtime	16
3.2.6	Client Errors	16
3.3	Assuring Client Server Connection	17
3.3.1	IP Takeover and NAT	17
3.3.2	DNS Updates	18
3.3.3	Several Configured Servers	19
3.3.4	Multiple 'A' Records	19
3.3.5	SRV Records	19
<b>4</b>	<b>Anycast, Replication and Authentication Design</b>	<b>21</b>
4.0.1	The Goal	21
4.1	Anycast	23
4.1.1	Stateless Proxies	24
4.1.2	Stateful Proxies	25
4.2	Sharing User Location Database	27
4.2.1	Client- vs. Server-based Replication	28
4.2.2	Trust Relationship	29
4.2.3	Network and Processor Load	30
4.2.4	Failed Replication	30
4.2.5	Cache Updates	31
4.2.6	Peer Status	31
4.2.7	Push vs. Pull	32
4.2.8	Keeping Removed Contacts	32
4.2.9	Delta vs. Full Transfer	33
4.2.10	Persistency and Synchronization	34
4.2.11	Multicast	37
4.2.12	Conclusion	37
4.3	Sharing Credentials	37
4.3.1	Trust the Location of the Registered Contact	38
4.3.2	Replicate H(A1)	39
4.3.3	Daily Pass for all Servers	39

4.3.4	Daily Pass for each Backup Server . . . . .	42
4.3.5	Distribute Precalculated Nonce-Response Pairs . . . . .	44
4.3.6	Public Keys as Password . . . . .	46
4.3.7	PK's with a Public Key Infrastructure . . . . .	46
4.3.8	PK's as Fallback . . . . .	47
4.3.9	Conclusion . . . . .	48
<b>5</b>	<b>Implementation of Replication and Authentication</b>	<b>50</b>
5.1	SIP-replication of User Location Database . . . . .	51
5.2	Distributed Precalculated Authentication . . . . .	55
<b>6</b>	<b>Validation</b>	<b>57</b>
<b>7</b>	<b>Conclusion</b>	<b>60</b>
<b>8</b>	<b>Outlook</b>	<b>62</b>
8.1	Load Distribution . . . . .	62
8.2	Peer-to-Peer . . . . .	63
	<b>Bibliography</b>	<b>65</b>
<b>A</b>	<b>Acronyms</b>	<b>68</b>
<b>B</b>	<b>Call Flow Messages</b>	<b>70</b>
B.1	Figure 4.1 . . . . .	70
B.2	Figure 4.2 . . . . .	71
B.3	Figure 4.3 . . . . .	74
B.4	Figure 4.4 . . . . .	76
B.5	Figure 4.5 . . . . .	78
B.6	Figure 4.6 . . . . .	80
B.7	Figure 4.7 . . . . .	85



# List of Figures

1.1	Broken Signaling prevents a Call . . . . .	3
1.2	The Fraunhofer Gesellschaft as an example of a federation . . . . .	4
3.1	Dependences of a SIP server . . . . .	15
3.2	Cause of Downtime [16] . . . . .	17
4.1	User updates contact list, requests current contact list or unregister a binding . . . . .	22
4.2	Successful SIP signaling through a backup proxy . . . . .	22
4.3	Split routing with Anycast and stateless proxies . . . . .	24
4.4	Non-straight signaling with the usage of Anycast . . . . .	26
4.5	Anycast with a failed stateful proxy . . . . .	27
4.6	Example SIP replication with backup failure and recovery . . . . .	35
4.7	Registration at a backup with SIP replication . . . . .	36
4.8	UA receives Daily-Pass from primary and uses it at a backup . . . . .	41
4.9	Difference between keys for each server and for each connection . . . . .	43
5.2	The state of a contact without the zombie state . . . . .	51
5.1	Design of the ULD in Memory . . . . .	51
5.3	The state of a contact after introduction of the zombie state . . . . .	52
5.4	Decision tree in the ULD timer event after zombie and replication introduction . . . . .	53
5.5	States of a contact within replication . . . . .	54
6.1	Replication statistics before test start . . . . .	58
6.2	Replication test runs normally . . . . .	58
6.3	Second peer failed during test . . . . .	58
6.4	Second peer still not recovered . . . . .	59

6.5	Recovery of the second peer detected . . . . .	59
6.6	All peers are synchronized again . . . . .	59
7.1	An example of a state-of-the-art solution . . . . .	61
7.2	A federation with SIP-replication improves availability . . . . .	61

# List of Tables

1.1	Measuring Availability [16]	2
1.2	Cost of Downtime [16]	2
3.1	Causes for Internet outages [15]	16
4.1	Replication alternatives comparison	33
4.2	Trusted location	38
4.3	Replicate H(A1)	39
4.4	One Daily Pass for all servers	42
4.5	Daily Passes for each server	44
4.6	Precalculated responses	46
4.7	PK's without PKI	47
4.8	PK's with PKI	48
4.9	PK fallback	48
4.10	Authentication schemes overview	49



# Chapter 1

## Introduction

The Session Initiation Protocol (SIP) is becoming the major signaling protocol for Internet-based communication. It fulfills the demands included in Integrated Services Digital Networks (ISDN) decades ago because it achieves real service integration and offers proven interoperability. SIP easily integrates the existing technologies of the Internet with instant messaging, presence services, voicemail and email, and network games. As SIP's adoption grows, new applications leveraging SIP's flexible infrastructure are likely to emerge. Adoption of SIP by 3GPP, the standardization body for next generation wireless networks, is yet more evidence of the success of SIP technology.

Nevertheless, SIP is still very new in 2003. Work on SIP began back in 1995 and the first mature SIP RFC that fixed the shortcomings of the previous version (2543) was published in 2002 [36]. Large network deployments are still taking off and many operational issues are still ahead of us.

One of the most challenging operational issues is that of fail-over performance. The bar has been set high by PSTN networks, which offer 99.999 percent reliability and impressively short fail-over times.

The goal of this thesis is to document the pieces missing from the SIP land-scape to achieve a fail-over performance comparable to PSTN. We will focus on issues related to SIP signaling. The entire problem breaks down into a couple of sub-problems that need to be dealt with.

### 1.1 Motivation

With the upcoming competition between PSTN and Voice over IP (VoIP), the still very new SIP protocol has to close the existing gap with the very mature PSTN technology. Certainly there are several regions in the competition where the SIP industry has to catch up with the PSTN. This thesis presents some concepts and implementations to catch up in the area of the signaling reliability.

As, for example, the analysis from Jiang and Schulzrinne [25] shows, the actual availability of VoIP on the Internet in general only reaches 98 percent. As you can see from

Percentage Uptime	Percentage Downtime	Downtime per Year	Downtime per Week
98%	2%	7.3 days	3 hours, 22 minutes
99%	1%	3.65 days	1 hour, 41 minutes
99.8%	0.2%	17 hours, 30 minutes	20 minutes, 10 seconds
99.9%	0.1%	8 hours, 45 minutes	10 minutes, 5 seconds
99.99%	0.01%	52.5 minutes	1 minute
99.999%	0.001%	5.25 minutes	6 seconds
99.9999%	0.0001%	31.5 seconds	0.6 seconds

Table 1.1: Measuring Availability [16]

Industry	Business Operation	Average Downtime Cost Per Hour
Financial	Brokerage operations	6.45\$ M
Financial	Credit card/sales authorization	2.6\$ M
Media	Pay per view TV	150\$ K
Retail	Home shopping (TV)	113\$ K
Retail	Home catalog sales	90\$ K
Transportation	Airline reservations	89.5\$ K
Media	Telephone ticket sales	69\$ K
Transportation	Packet shipping	28\$ K
Finance	ATM fees	14.5\$ K

Table 1.2: Cost of Downtime [16]

Table 1.1, this means an outage of 7.3 days per year, or 3 hours and 22 minutes per week. The availability of PSTN depends on many factors, for instance, on the type of customer, but can reach 99.999 percent. Downtime does not only cause inconvenience to the callers and the callee's when they cannot reach the desired callee, or even worse, cannot place any call. It also means a loss of business opportunities and, depending on the business, can be very expensive, as you can see in Table 1.2.

If the goal is that VoIP over a computer network makes obsolete or replaces an extra network for PSTN phones, it also has to serve emergency calls. And if human lives are depending on the availability of the SIP network, nobody would want to account for the unavailability of the phone service for several hours.

One of the fundamental characteristics of VoIP with SIP is that signaling with SIP usually takes a completely different path through the Internet than the media of a running call. Thus it can happen that a call between two participants cannot be established only because one of the servers in the signaling chain is not reachable. Figure 1.1 illustrates the problem. That one of the signaling servers in the "SIP trapezoid" [36] is not reachable can be caused by software or hardware failure, by the failure of a required service, by catastrophes in the environment of the server, by problems in the IP connectivity or

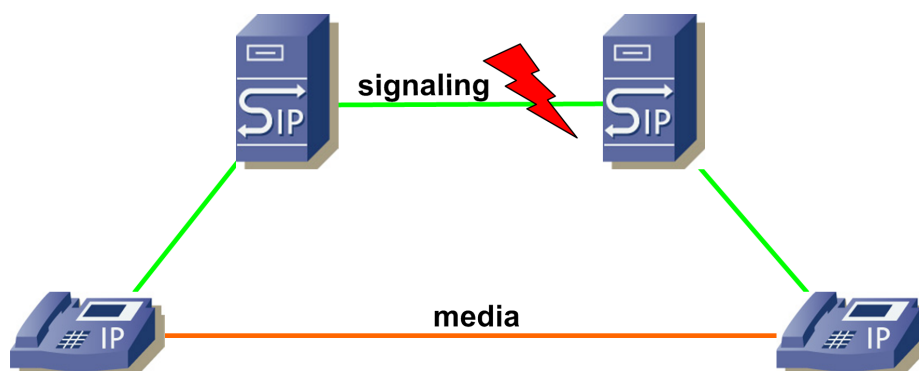


Figure 1.1: Broken Signaling prevents a Call

malicious attacks on a server, or at last simply by planned downtime. Therefore we will try to improve the availability of all the signaling paths with this work.

## 1.2 Objectives

We present here a solution to protect a SIP service against all error types except client failures. We join the forces of several geographically distributed SIP servers to a community which we call a federation. All clients in this solution have a primary server which they normally use. But if their primary server is not reachable for one of the clients, it can simply switch over to one of the other servers within the federation. The federation solution is cheaper than the classic High Availability (HA) solutions, portable and not bound to the use of specific technologies or products. This should reduce the Total Cost of Ownership (TCO) for a highly available SIP service, although we do not investigate this perspective deeper in this thesis.

The goal of our federation architecture is to provide a solution similar to the well-known mail (SMTP) systems. For the SMTP system the information, which hosts are primary and backup servers for the domain, is distributed over the Domain Name System (DNS) system. The sending side (client) tries to deliver the mail to the specified backup hosts if the primary server<sup>1</sup> does not respond. These backup hosts can be located at very different locations than the primary server, and they just have to be configured to accept mail for the domain for which they are referenced as backup hosts.

And this is also the target of our SIP federations approach, that the clients should simply switch over and the backup server should only be configured to accept the traffic for the other domains, but with as little additional administration effort as possible.

The institutes of the Fraunhofer Gesellschaft in Germany are a good example of a federation. The institutes are geographically distributed over Germany and each has its own computer network with separate domain and administration. Figure 1.2 shows that *sip:alice@iptel.org* could use any of the three servers of the example federation to call *sip:bob@fokus.fhg.de*. The server *sip.fokus.fhg.de* would be the

<sup>1</sup>Primary mail server is the server with the lowest priority out of the list of possible servers.

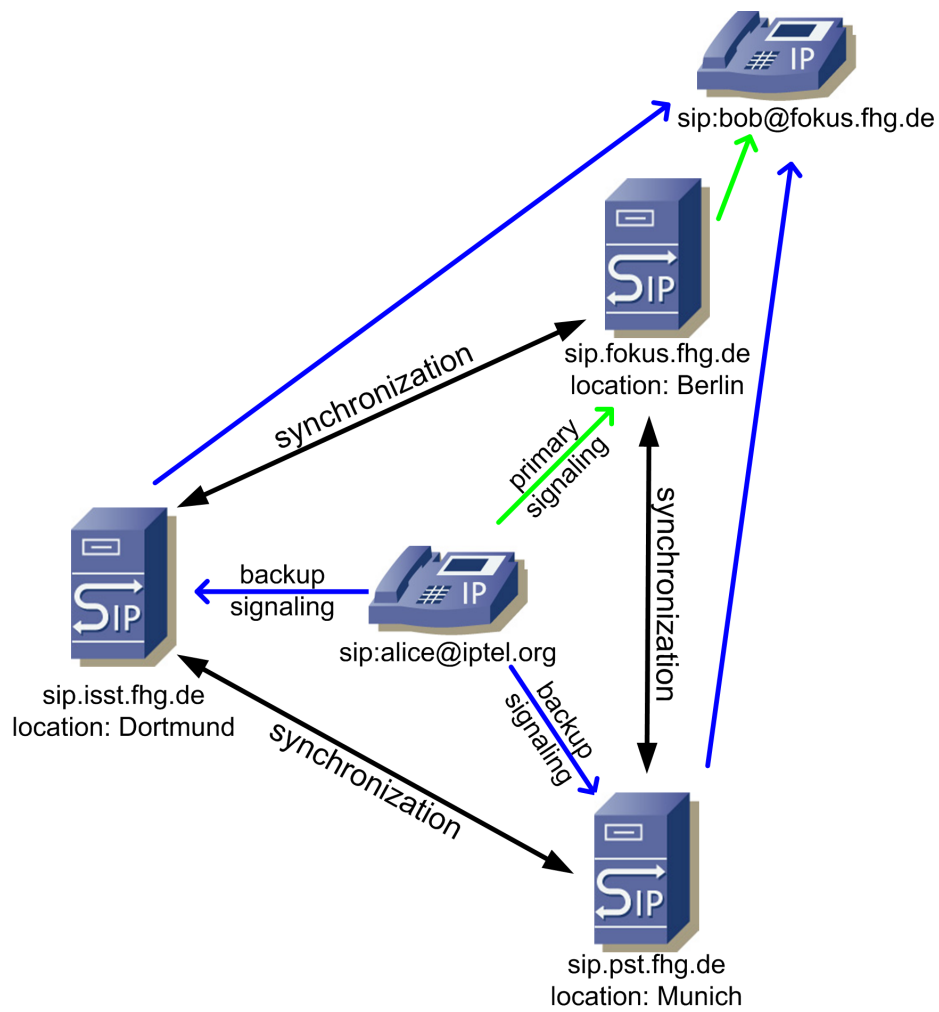


Figure 1.2: The Fraunhofer Gesellschaft as an example of a federation

primary server for Bob, but during a failure of this server any of the other servers could be used to locate Bob. And Bob can use any of the other servers to call Alice during the outage of its primary server.

If we compared a SMTP and a SIP federation there are some differences. The mail delivery does not have to happen in real time and the chain of mail servers is stateless, because each host in the chain from source to destination can delete any references to the mail after it delivered it successfully to the next hop. In contrast everything in SIP has to happen in real time. Even though the protocol is designed to store as little state as possible it needs some. Most of the required state is kept in the clients, the end points of the signaling, but the servers, for example, have to store where a user can be reached to allow user mobility.

The servers within the federation will share the information where a user currently is located, which is called the user location database, and thus guarantee the availability of each user within a federation as long as at least one server from the federation is reachable. In some cases, the servers within a federation may not trust each other enough to share the passwords of their users because they are only loosely interconnected and only help each other in case of failure. Additionally it would require administrative effort to keep a distributed password database synchronized. Thus we will present different solutions which avoid the sharing of the passwords of all users, but nevertheless allows the servers to authenticate the users.

The presented solution is implemented in two add-on modules for iptel.org's SIP Express Router (SER).

### 1.3 Scope

The scope of this thesis covers only the availability of SIP, which means it covers only the improvement in the availability of the signaling in comparison with PSTN. We do not look at how to improve the availability of a host or a single server. This means we do not look at improving the reliability of IP, of required services like DNS, databases or anything similar and protection against hardware failures. In addition, improving the packet loss ratio of VoIP connections or the reliability of established VoIP connections is not considered here, because the reliability of the media does not belong to the signaling part.

Naturally, the path between the two phones in Figure 1.1 can also be disturbed and prevent a successful call between the two participants. Or one of the participating UA's can crash during the call. To prevent this, we would have to either improve the availability of IP and routes or improve the reliability for the UA's, but that is not in the scope of this thesis. For improving the reliability of already established calls, we refer to [35].

### 1.4 Document Structure

Chapter 2 gives information about how High Availability is achieved nowadays on the Internet and in SIP networks. Chapter 3 analyses the requirements to achieve a HA SIP service. In chapter 4 we first explain the required changes to Anycast for SIP

servers, then discuss how we can assure the reachability of the users, and finally how authentication can be solved in a federation. Details about the implementation can be found in chapter 5. The chapters 6 and 7 validate what we achieved and give a final summary. What is left as future work is described in chapter 8.

## Chapter 2

# State of the Art

Firstly, we will present what is possible in PSTN. Secondly, we will state the currently used techniques for High Availability on the Internet and in local networks. And finally, we will explain what is used to achieve HA in Voice over IP networks and for authentication.

### 2.1 PSTN

In the area of the reliability, or more precisely availability, of PSTN, several sources mention the famous five nines (99.999 percent) while others range from four (99.99 percent) up to six nines (99.9999 percent) [25, 4, 26, 32]. AT&T, for example, claimed to have achieved 99.98 percent availability in 1997 [3], but without any explanation as to how and from which data exactly this value was calculated. In general, these many nines seem to be only valid for the backbones of the big carriers or for big business customers, and not for the private customers of the carriers. Some private conversations and [4] also evinced that the famous nines are mostly only numbers from the specification of the PSTN switches and thus the over all availability does not have to be the same. Unfortunately we were not able to find any source which documents how and from which statistics these numbers were calculated. Although it is possible to get statistics of all the big US carriers from the FCC [10], it is beyond the scope of this thesis to analyze and interpret this mass of detailed information.

### 2.2 Internet

The High Availability of the Internet is marked by redundancy. We will briefly present what else besides redundancy can be used to improve availability.

#### 2.2.1 Redundancy and Cluster

Today High Availability is typically achieved by installing the hardware and the supplying environment redundant. This can only be done for specific components like the

network interface card or the power supply. If the service should be protected against all possible hardware failures within a host, a complete second host will be installed as redundancy. Because of the performance improvements and the low prices of modern PCs, many hosts will often be installed as a cluster to achieve High Availability. Besides the possible load distribution between the hosts of the cluster, this also has the advantage that more than one host may fail before the service is interrupted. But a redundant host also does not protect against environmental failures like power or Internet connectivity failures, so the whole supply environment has to be made redundant as well. And even this does not protect against natural catastrophes like earthquakes, blizzards or anything similar which may paralyze the whole area [16].

### **2.2.2 IP as Single Point of Failure**

If the serving host, its computer center and the whole supply environment is redundant, the IP address of the service will still remain as a Single Point of Failure (SPoF). This is because if the name of the service host resolves to only one IP, then anything in the networks between the client host and the server can interrupt the service. For service providers on the Internet it is very unlikely that they can control the whole path between client and server.

### **2.2.3 DNS**

A way around the single IP as SPoF could be to resolve the name to more than one IP, or to use DNS Round Robin. But if the client chooses or gets the IP out of a set which is not reachable, the client will probably fail. Another solution for the case of a failure of one of the servers out of the set is to remove the IP of the failed server from the DNS. However, this requires an observing unit which updates the DNS on a failure of one of the servers, and because of the caching nature of the DNS system, such DNS updates can take days until they are visible to the client.

### **2.2.4 Multicast**

In many publications about replication or fault tolerance Multicast is proposed for communication between multiple entities. Atomic Multicasting in particular does make sense to save bandwidth and resources at the participating entities [27, 41, 33, 2, 31]. Unfortunately multicast needs the explicit support of all routers in the Internet between all participating hosts of a Multicast group. And as TCP is not possible with Multicast special new transport protocols have to be used if reliability is required at the transport layer. Thus the use of Multicast is restricted to LAN's or specifically prepared networks only.

### **2.2.5 Anycast**

Another possible solution is Anycast (RFC1546 [6]), where a router forwards an incoming packet to one of the servers. If the router is able to remove the failed server out of the server set for a given Anycast address, this will help only available servers to be

used. Another advantage is that the client uses, measured in router metrics like number of hops, the nearest and therefore probably the fastest server. This solution may also have several drawbacks, depending on the implementation. As described in [37], Anycast is already used in the IPv4 environment of the f.root DNS server, but it is only used in the internal WAN to distribute the DNS queries to the running servers. From and to the Internet the f.root servers have only one connection in Palo Alto, which means that the clients from the Internet do not really use and do not benefit from the nearest server feature of Anycast. For a more detailed discussion of Anycast and its use for SIP see Section 4.1.

## 2.2.6 rserpool

The Reliable Server Pooling (rserpool) working group at the IETF has tried to develop an architecture which provides a pool of servers for a service, which can be dynamically managed and so should improve the availability of the service. Basically, the architecture consists of some protocols which enable a client to query for the addresses of servers which can currently provide the desired service. Additionally the new rserpool name server is able to manage the members of the service dynamically. So ultimately the rserpool architecture is a replacement for the DNS system with a better management capability to react to dynamic changes in the relation between query and answer. In addition, rserpool provides an optional feature which suggests to the client alternative servers which the client could use on the failure of the currently used server. But rserpool does not explain how a client should determine if and when a currently used server has failed. Also rserpool does not guarantee that the client is able to reach, on the IP layer, one or all of the servers in the pool and the answer to the service request. It should also be mentioned that at present the rserpool protocols are not designed to work on the whole Internet namespace, but only on a small subset like in a local network. To summarize, rserpool adds some useful features to the DNS system which may also be solvable without any new protocols, but it does not solve any problems we have with current solutions either.

## 2.3 SIP / VoIP

We presume the knowledge of the SIP protocol within this thesis and refer to chapter 2 of [24] for a good introduction to SIP, and to RFC3261 [36] for all technical details.

### 2.3.1 SIP Availability Today

Unfortunately there is not much material available about the reliability of SIP and VoIP. Jiang and Schulzrinne [25] set up a test network of test clients in many different networks and simulated automatically random calls between these clients. The result of measurements of packet loss ratio, call success and abortion probability is the "overall service availability" of 98 percent. Although the 98 percent already includes the probability of media errors, the figures of 99.53 percent availability just for the signaling part is also not realistic in the case of SIP. This is because in the test environment the clients signaled the connections directly without any proxies, but in a realistic SIP use case

there is at least one, normally two or more proxies used (one as outgoing proxy and one as incoming proxy for a call from one domain to another). The typical SIP use case is the SIP "trapezoid" shown in Figure 1.1. This results in a more realistic availability of 98.51 percent ( $0.995^3 = 0.985074875$ ) for a call through the "trapezoid", or 98.01 percent ( $0.995^4 = 0.9801495006$ ) if signaling errors directly between the clients are also included. In addition, for the overall network availability the 1.5 percent call abortion probability has to be subtracted from this value. But this call abortion probability is not investigated here any further, which means that we do not specifically look at the case of whether a call was established successfully between two users but then aborted because of some error in the link between the two UA's. It should be no problem to negotiate another new connection between the two UA's, but it is questionable if this second call can be established successfully if the error on the direct link between the two clients still persists and was not only temporary.

### 2.3.2 Intelligence in the Network

As described in [8], the availability of the signaling path can be improved by placing the intelligence to detect an error and react on it in either the previous hop or in a redundant hop. The intelligence in the redundant hop requires support for a special routing protocol like Virtual Router Redundancy Protocol (VRRP) [29] or Hot-Standby Routing Protocol (HSRP), which cannot be guaranteed for all hops in the Internet or only do work for local networks. Thus we are convinced that the intelligence should be placed in the previous hop, even though this strategy is probably not followed in the majority of existing solutions.

The usage of "DNS RR for specifying the location of services (DNS SRV)" [1] is one possible way to place the intelligence in the previous hop. RFC3261 references this technology for use in all SIP implementations. DNS SRV will be discussed in more details in Section 3.3.5.

RFC3261 includes a description of the usage of Multicast with the SIP protocol. But the main focus lies in the support of Multicast for registering at the next unknown registrar and for the usage of Multicast as transport for conferencing sessions. It does not include support for replication or fault tolerance, and we are not aware of any SIP implementation which uses Multicast.

### 2.3.3 Other SIP Products

The information about how other available products try to solve the problem of availability is mostly not publicly available and relies on personal information and communications.

#### Cisco

Cisco seems to use a proprietary protocol to replicate information from the registrar to other hosts.

## **dynamicsoft**

Dynamicsoft relies completely on the usage of the replication features of the Oracle database connected over the JDBC interface. It is obvious that the product has to check every request against the database and can not use any cache if a database replication is used. (Section 4.2.5 explains the details about replication and caches.)

## **Ubiquity**

Ubiquity claims to achieve scalability and fault tolerance with the usage of redundancy. They use a load balancer, called Service Director, to distribute incoming requests to the hosts of a cluster. The service hosts are connected over JDBC to a database. This database is probably used to replicate respectively make the information for all hosts available. [11]

## **Vovida**

Vocal, the SIP server from Vovida, seems to be able to use some kind of SIP replication. But it has been reported that the product has problems with synchronization after a failure recovery because of the speed of the incoming "update" messages.

### **2.3.4 H.323**

The H.323<sup>1</sup> standard from the ITU-T defines backup hosts in their call setup procedure. In H.323v5 [21], it is defined that a device gets to know its gatekeeper either by manual configuration or by sending a special request to a well-known Multicast address. Willing gatekeepers should answer this request, and in this response they can also include a backup host for their service. So the service reachability from the perspective of a client relies on a working Multicast network, or that the manually "hard" configured gatekeeper is reachable on the IP layer. In H.323 Annex R [20] and H.323v5 two new methods are defined which make a failover for a running call possible. But these new methods are only necessary because H.323 is call stateful<sup>2</sup>, which a SIP proxy is normally not. Also, the availability of H.323 gateways and gatekeepers seems to be only secured by clusters. A standard or procedure as to what a client can do if it does not discover a gatekeeper automatically or cannot reach its configured gatekeeper is not defined.

### **2.3.5 SIP Digest Authentication**

Because authentication is also a big part of this thesis, we will first explain the authentication technique currently used for SIP.

---

<sup>1</sup>H.323 is a collection of binary protocols for IP-based video conferencing and telephony defined by the International Telecommunication Union (ITU).

<sup>2</sup>Call stateful means a signaling device stores the state of each call from the start to its end.

The authentication technique for SIP is based upon the HTTP Digest Access Authentication (DigestAuth), since RFC3261 [36] that has been RFC2617[18]. But RFC3261 forbids the use of Basic Authentication for SIP, which is also described in RFC2617.

As a simple overview the DigestAuth scheme works like this: for every request for which a UAS wants an authentication of the sender, and with no authorization header already present, it sends back a 401 *Unauthorized*, which contains a Proxy-Authenticate or a WWW-Authenticate header. This authenticate header contains a challenge, the nonce, which the client uses to generate a response, with this nonce, his password and some other fields. This response is put into a Proxy-Authorization or Authorization header. The client then includes this header into the request, starts a new transaction with this new request and sends the new request. If the UAS is able to calculate the same response, the user is authenticated.

The UAS is free to create the challenge with any algorithm. A SIP servers typically include some of the headers of the request, e.g. the Contact header, and a server secret to create the nonce. It also often includes some kind of timestamp, so that the server is able to check if the nonce is out of date.

The UAC first has to calculate the hash sum, normally with the MD5 algorithm, of the A1 string. The A1 string consists of the name of the user, which means the username for this authentication and not, for example, the real name of the user. The second part is the realm value, which the UAS should provide in the authentication header and which should help the user with what he should authenticate so that he can give the correct username and password. The third and last part of the A1 string is the password of the user.

$$A1 = \text{username} : \text{realm value} : \text{password}$$
$$H(A1) = \text{Hash}(A1)$$

Because all the values of the A1 string are independent of the transaction, the hash of A1  $H(A1)$  can be precalculated and stored on both sides (client and server) to save some CPU cycles during the authentication.

Secondly the hash of A2 is required. The A2 string consists of the SIP request method, e.g. *INVITE*, and the request URI as SIP-URI or SIPS-URI.

$$A2 = \text{method} : \text{request URI}$$
$$H(A2) = \text{Hash}(A2)$$

Because of backward compatibility with RFC2069, which was used in RFC2543, it is possible that a UAC or UAS does not support "quality of protection" (qop) parameter. In this case the response value is calculated like this:

$$\text{response} = H( H(A1) : \text{nonce} : H(A2) )$$

However, all RFC3261 compliant implementations have to support qop. In this case, the value(s) of qop indicates if the UAS only supports authentication protection ("auth") or also authentication with integrity protection of the message body ("auth-int"). The UAC

can select the best supported method out of this list. In both cases, the UAC creates a random string, the nonce, to protect the request against chosen plaintext attacks, for mutual authentication and, in the case of "auth-int", for message integrity protection. Additionally the UAC needs to count how many requests, or more precisely transactions, it has answered with this nonce and provide this value in the nonce-count field. If the UAS stores the last seen nonce-count value for each client and compares it with the current value from the request, it can detect if a request is sent in a replay attack.

If "auth-int" is used as qop, then the A2 string additionally includes the hash of the request body to protect the integrity of the message body. In the absence of a request body, for example in REGISTER requests, it is calculated as hash of "". So the A2 for "auth-int" looks like this:

$$A2 = \text{method} : \text{request URI} : H(\text{request body})$$

Finally, if qop is used the response is calculated like this:

$$\text{response} = H( H(A1) : \text{nonce} : \text{nonce-count value} : \text{qop value} : H(A2) )$$

In any case, as input for another hash calculation as well as final output for a header, the output of the hash function is converted into hexadecimal representation before proceeding.

## Chapter 3

# Requirements for HA SIP services

The requirements for making a SIP service highly available are complex, and therefore we will analyse them in this chapter first before we design the service in the next chapter. For this requirements analysis and the design we assume a SIP service which provides its service over the Internet. The requirements and design of the same service for a local network do not have to take into account the problems of the long connection paths of the Internet, but the requirements of an Internet service imply all the problems of a local service.

Firstly, we show up the dependencies of a SIP service and explain what is mandatory for such a service. Then we explain the possible error causes and their impact, and finally we analyse what methods are best suited for a SIP service to assure the connection between the client and the server.

### 3.1 SIP Dependences

As already noted in Section 1.3, this thesis only covers how to improve the reliability of signaling. Figure 3.1 illustrates in the "depends" relation of fault-tolerant systems [12] that a SIP server depends on many services to work correctly. A SIP server in this case means a *SIP Proxy* with location lookup and registration functionality. The use of DNS names in a lot of the SIP headers introduces more flexibility but is also a bottleneck on performance [24], and makes a SIP server fundamentally dependent on a reliable and fast *DNS* service. And the *Proxy* itself, the *Registrar* and the *DNS* service depend on a working network connection too. *Authentication* can also depend on a working network connection if the authentication, for example, uses a Radius or LDAP server, but we assume that authentication normally does not depend on network connection. A *SIP Registrar* normally stores the contacts of a user on a reliable and persistent *Storage* service, which could be anything from a plain text file up to an HA database service. The *Storage* of the contacts is not mandatory and could also be just to hold the memory of the *Registrar*. But if the *SIP Proxy* or the *Registrar* wants to

authenticate a request from a user it relies on a service which can provide the password of the user from a reliable and persistent storage. If a database service on a different host than the *Registrar* is used for this purpose, then the storage service would also depend on a working network connection.

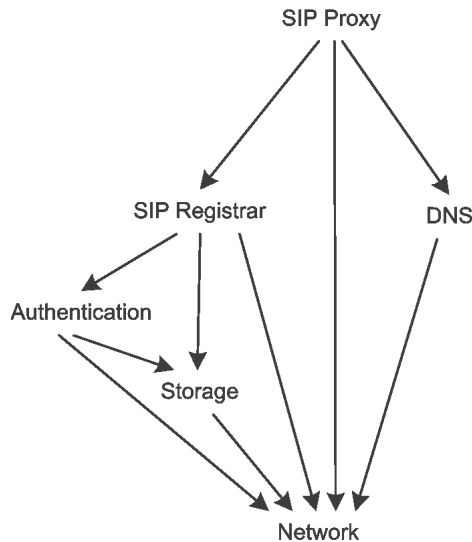


Figure 3.1: Dependences of a SIP server

We do not investigate how to further improve the availability of the required services of a SIP server. It should be clear now that it is hard to protect a SIP server with all its depending services against a failure. But with a geographically distributed federation of servers the probability that all servers from one federation will depend on the same instance as of one of the services should be close to zero. The weakest point within a federation is that the clients have to be able to resolve the domain name and to receive the SRV records of the federation. Thus it is an absolute requirement for this approach that there are several DNS servers in different networks which are able to resolve the domain of the SIP service.

## 3.2 Error Types

The causes for the unavailability of a service are multifarious. We will give here a short overview of the types of possible errors and their impact.

### 3.2.1 Software, Hardware and Required Services

At first there is the failure of the server software and it goes further with the defect of a part of the server hardware. Errors in required services like the Domain Name System, Databases or Network File Systems can also cause the failure of a superior service. All these error sources may be eliminated by installing them redundantly. This redundancy minimizes the outage time of a service and all its dependents, and is the easiest solution to assure the availability of a service.

### 3.2.2 Environment

Protection against the abnormal behavior of the host's environment is a lot harder. Power outages can be bypassed with a standby set, and air conditioning protects the hosts against overheating. Protection against threats of fire and water should be possible. Besides the fact that such protection is very expensive, it is nearly impossible to guarantee that a service will still be running normally after such a catastrophe.

Maintenance	16.2%
Loss of power	16%
Fiber cuts	15.3%
Hardware failure	9%
Routing errors	6.1%
Congestion	4.6%
Malicious attack	1.5%
Software bug	1.3%

Table 3.1: Causes for Internet outages [15]

### 3.2.3 IP-Connectivity

Not only an environmental catastrophe must happen for a service to become unavailable to its customers. An error in the local network, like broken or removed cables, failed network cards, router and switches, may be quickly repaired by the service provider if it is detected, or even better eliminated or minimized by redundant networks. A redundant uplink to the Internet over two or more different Internet Service Providers (ISP) is also possible and protects against the failure of one of the providers. But if the network of one of the carriers between the customer and the service provider does not work normally, the service provider cannot often repair this easily. Examples of this are routing errors, congested links, outages of non-redundant parts of the Internet or the time until an updated routing entry takes effect.

### 3.2.4 Malicious Attacks

Everybody knows since the attacks in February 2000 on Yahoo, CNN.com, Amazon.com, eBay.com and many other sites [9, 39] that Distributed Denial of Service (DDoS) attacks can bring down even major services for hours. At least for the customers it makes no big difference if the service is brought down by an attack or they can not use service just because of the congestion produced by the DDoS attack itself. And protection especially against DDoS attacks with spoofed source addresses is not that easy [39].

### 3.2.5 Planned Downtime

A significant fraction of service outages is caused by planned downtime, as we see from Figure 3.2 and Table 3.1. However, this part is normally relatively easy to protect because a switch over to another host can be prepared in advance, although this is another argument for a highly available environment, because it reduces the preparation for the downtime and makes short-term downtimes possible.

### 3.2.6 Client Errors

The last type of error lies on the client side and its surroundings. This includes errors in the client software, failure of the client hardware, errors at the uplink of the client

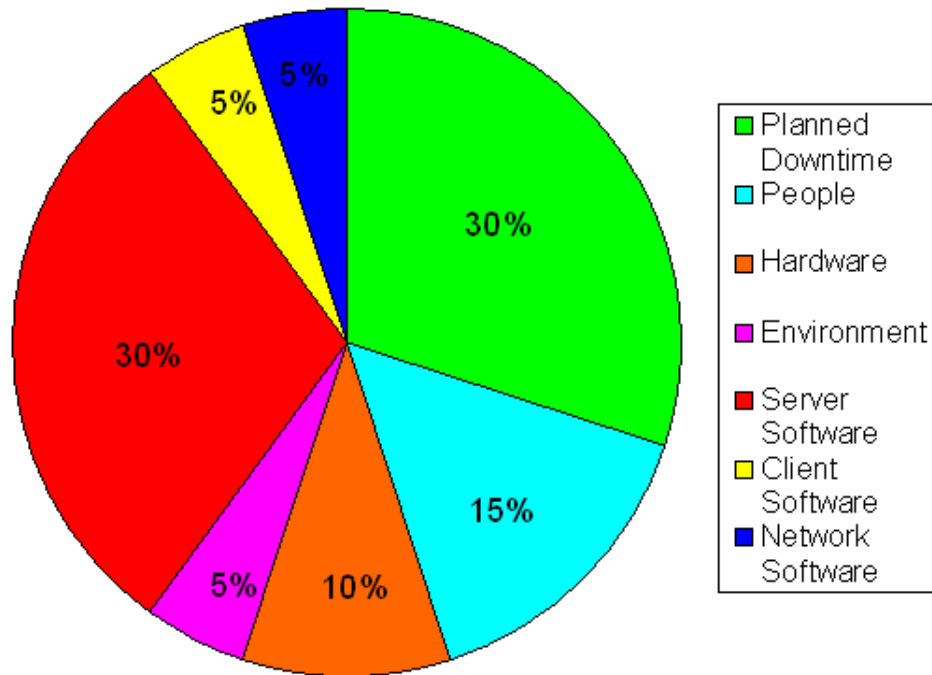


Figure 3.2: Cause of Downtime [16]

and all the other error types which can happen to the server as already described. For a service provider which offers its service over the Internet these types of errors are typically out of the control and have to be solved by the user itself. In some companies it may also be necessary to provide highly available clients to some users, for example, with two clients to each user. But usually it should be sufficient to replace a failed client in a given amount of time or repair it directly.

### 3.3 Assuring Client Server Connection

One of the most significant problems in making a service highly available is to ensure that clients are able to find one or more running servers at any time. Client and server means in this case User Agent Client (UAC) and User Agent Server (UAS), as described in RFC3261 [36]. Various solutions to this problem exist with different advantages and disadvantages. We will now briefly discuss these solutions and conclude which are the best for our high availability SIP server architecture.

#### 3.3.1 IP Takeover and NAT

The best known and most widely deployed solution is IP takeover. This method, for example, is also used with the VRRP [29] protocol. In this case, one unit observes the server, and if it fails, the observing unit shuts down the running server and starts up

a backup server. The backup server uses a network interface with the same IP as the previous server (IP takeover). It does not matter if the observation of the running server is done by the backup server itself or by a third unit.

The transfer of the IP address raises some problems in the case of Ethernet, because the association between the IP address and the hardware interface MAC address is cached for some time at the other hosts in the same network. To prevent the other hosts in the network from sending requests to the old server, the backup server will start to send gracious ARP responses for some time to propagate the new IP MAC association. Another solution is for the backup server to use a network interface card with the same MAC address as the previous server. Besides the fact that it does not conform to the Ethernet standard IEEE 802.3, this might raise a problem with hardware Ethernet switches. Because Ethernet switches store which MAC address is reachable at which ports of the switch, the switch would deliver the packets to the wrong port if the backup server was not connected to the same port as the previous server. Whichever of the MAC address problem solutions is chosen, the IP takeover solution has the big disadvantage that it can only be used in the same network, except where an update of the routing tables has been considered (see below for this solution).

An alternative to IP takeover, but with the same drawbacks, is a solution which uses Network Address Translation (NAT) to switch the traffic over from the failed server to the backup server. In this case, the host with the destination IP is not running the real service, but changes the destination IP of the incoming packets (DNAT) and the source IP of the outgoing packets (SNAT). In the case of a failure of the running server, the NAT host would replace the IP of the running server with the IP of the backup server in its NAT tables. One advantage is that the running and the backup servers do not have to be in the same network, because the node which makes the NAT can send the packets to any other node in the public Internet or in a private network. If it is used in the public Internet only, the NAT is only a replacement to update the routing tables. But with this solution, the problem is only transferred from the hosts which run the service to the host which makes the NAT. This is because if the NAT host fails, another host will have to take over the NAT “service”, and this takeover raises the same problems as the normal IP takeover does.

### **3.3.2 DNS Updates**

A second solution is the DNS update. In the case of the failure of the running server, a monitoring unit would change the association between the DNS name of the service and the IP of the failed server to the IP of a backup server. This solution is very simple but has the big drawback that DNS responses are usually cached for some time. It is possible to define a short caching time for the domain name of the service. But firstly, a short DNS caching time produces a lot more network traffic for the domain. And secondly, not all implementations respect the given caching time. Finally, the biggest refutation is that we have seen some SIP UA implementations which only resolve the DNS entries of their configuration one time at their startup. These UA’s would never recognize an updated DNS entry.

For the following solutions, the client has to detect that a connection to the server is not possible or has got lost. This detection is normally based upon time-outs, which signal the client that the server has not responded in a suitable way in a given amount of time.

A combination of timeouts with the correct interpretation of ICMP error messages can improve the client reaction time to connection loss with the server dramatically. This is because if the client understands an ICMP message from the server or the network environment correctly, it does not have to wait for the timeout for this connection before it switches to another server. Thus client-based connection lost detection gives the best results for each client, because only the client itself can safely decide which server is reachable or not.

### **3.3.3 Several Configured Servers**

Probably the most obvious solution is to configure multiple servers at the client. Depending on the implementation, the client will then use the configured servers in a round robin scheme, or switch from one server to another after the connection to the current server is lost. But this alternative has many drawbacks: the number of servers, the switching algorithm used and whether some of the servers can be used with a priority over the other servers all depends on the implementation. And if one of the server addresses changes, the administration has to change the address at every client, although this may be eased by system-wide configuration of the client. But if the server provider does not have administrative access to the clients, it can only inform their users about the address change and advise them to change the configuration of the client. So to configure multiple servers at the client is the most inflexible client-based alternative.

### **3.3.4 Multiple 'A' Records**

The next alternative is to configure multiple A records in the DNS system for the DNS name of the server. This has the advantage that servers can easily be added to or removed from the set of records by the server administration. If this is used to remove failed servers from the server set, then this solution will inherit the attributes from the DNS update described above in Section 3.3.2. But this alternative relies on the handling of multiple A records by the client. It can happen that a client uses all servers in the set in a round robin way. Some clients use one of the servers until the connection to this server is lost and then switch to the next IP address in the set. And in the worst case, a client may only use one of the IP addresses all the time. Although this solution is very easy to administrate, the undetermined behavior of clients makes this alternative not very attractive.

### **3.3.5 SRV Records**

Probably the most flexible and modern approach are DNS SRV records[1]. With this solution, multiple servers with different priorities can be specified for every service type. With the priority values it can clearly be specified in which sequence the client should try to connect the servers. Additionally, the weight value gives an easy option for load distribution and a non-standard port for every server can be specified. According to RFC3261 [36] and RFC3263 [23], all UA's should use DNS SRV records to look up the destination of the request. Unfortunately, our tests with some SIP UA's showed that only one of the UA's had a correct implementation of SRV support, with switch-over

after a server failure and successful switch back after the recovery of the failed server. But if the UA's support SRV correctly, then this is the easiest to administrate and the most flexible client-based solution to assure a connection between the client and one of the servers.

To sum up, the best solution to ensure that the client has a connection to a server is client-based<sup>1</sup>, because the client can best decide to which host connections are possible for it. And the best client-based solution is based upon the use of DNS SRV records, because of their flexibility and easy administration.

---

<sup>1</sup>Client-based means UAC in the terms of SIP, and is the previous hop, not only the initiator of a request.

## Chapter 4

# Anycast, Replication and Authentication Design

For the design, we will first draw a more detailed picture of what we want to achieve than is found in Section 1.2.

During our research we also considered the use of Anycast for an HA SIP federation, so we will present our solutions for the combination of SIP and Anycast in this chapter too.

The biggest part of the design of an HA SIP server architecture breaks down into two problems. The first part is how the user location database (ULD) can be replicated over the peers of a federation to guarantee or at least improve the reachability of the user. For the second part, how to solve authentication within the federation, also when the peers from the federation are not trustworthy enough to be given the passwords of the users, we will present various solutions.

### 4.0.1 The Goal

The call flows in Figures 4.1 and 4.2 illustrate how server and clients should ideally react to the failure of the primary server. (The example messages for each call flow within this thesis are located at Appendix B.)

The call flow in Figure 4.1 shows that the backup registrar accepts the contact change from the user too. The backup registrar should be able to challenge the user, like in Figure 4.1, although it does not know the password of the user. In Section 4.3 we will discuss how this can be achieved.

From the call flow in Figure 4.2 we can see that the backup proxy forwards the request to the correct destination too. In Section 4.2 we will discuss how this can be achieved.

The second issue from Figures 4.1 and 4.2 is that the sender detects that the primary server from the receiving side is not reachable anymore or does not respond, and that a backup server should be used. For details see Section 3.3 and [35].

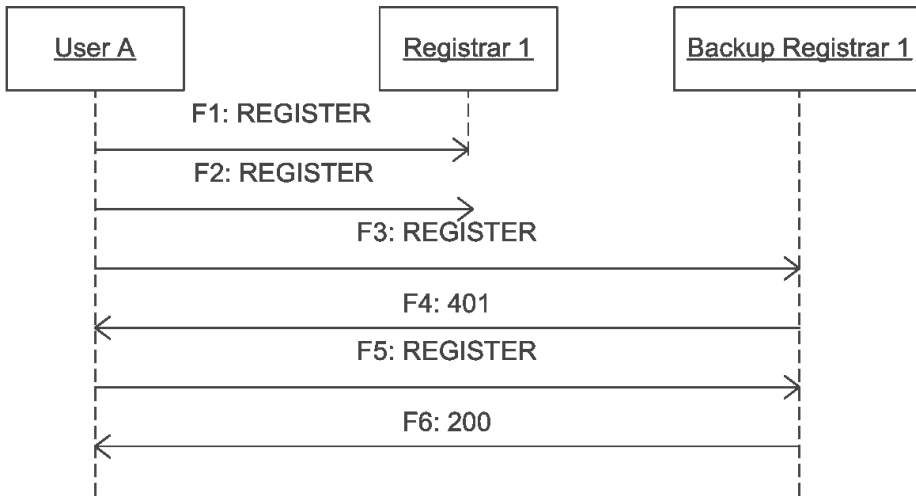


Figure 4.1: User updates contact list, requests current contact list or unregister a binding

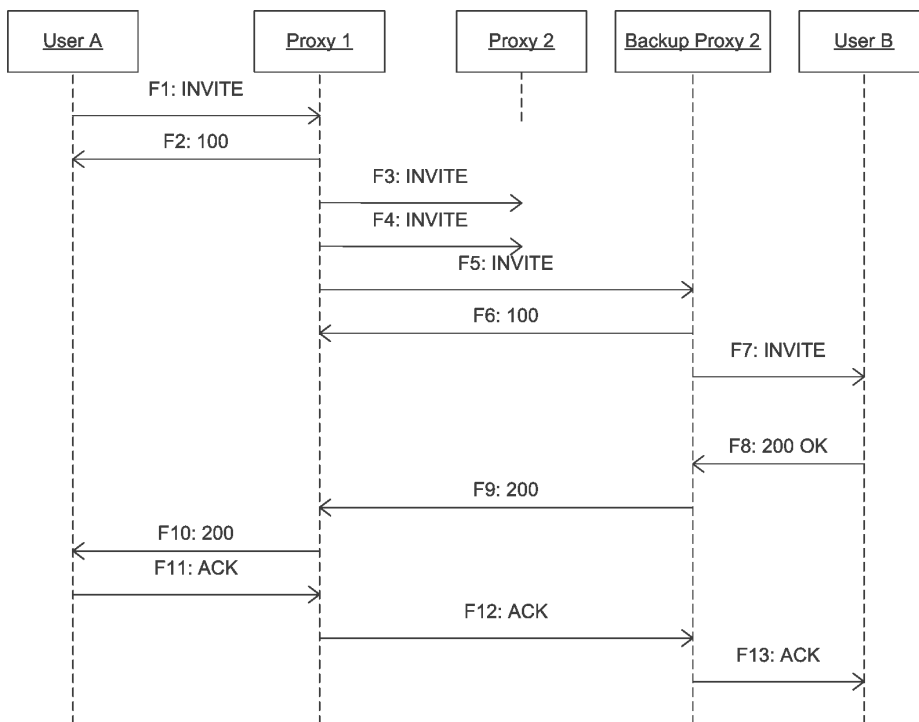


Figure 4.2: Successful SIP signaling through a backup proxy

Depending of the chosen HA solution it might happen that a proxy has to relay a response or acknowledgement to its destination, without seeing the initial request. Because of the design of the SIP protocol any server should be able to forward a response or acknowledge to its destination statelessly, without any special precautions. Problematic in this case are only Record-Route and Route headers if they contain IP's, because an IP can be a SPoF itself as explained above. But if domain names are used in the Record-Route and Route headers, there should be no problem in relaying requests or responses because the backup servers from the SRV record can be used. Thus we do not cover how to improve all parts of signaling, because the fallback to stateless relaying should always work. Also, the effort in CPU time and bandwidth to keep all backup servers synchronized about the states of all transactions is much higher then the benefit in the case of a small switchover time, where replies are only relayed statelessly.

In the terms of fault-tolerance we try to create a *group masking* solution with our federations approach where "individual member failures are entirely hidden from the users" [13, 12]. This means in our case that a user can still use the service although one of the servers is not reachable, but not because the federation as a group provides an answer. The UAC has to elect one of the federation members that is still working to send its request and get answers from this entity.

With this federations approach we cover *crash*, *omission*, and *late timing failures* of a server within the federation [12, 41, 13]. *Response failures*, with *value* and *state transition failures*, and *random failures* cannot be covered by this approach.

Of course, any of the methods described in Section 2.2 can be used to improve the availability of a single server. Today, for example, the implementations of the VRRP [29, 8] protocol are very popular to enable the IP takeover scenario. For all of these methods, the sharing of the user location database as described in Section 4.2 is still more or less a problem which has to be solved. If one of the local solutions is chosen, authentication as described in Section 4.3 may not become a problem. On the other hand, these local solutions do not protect against errors in the networks between the client and the server. Therefore the local solutions should only be used to protect against software and hardware failures, but should be combined with a geographically distributed federation.

## 4.1 Anycast

A more network-oriented solution to protect also against errors in the networks between the client and the server is to update the routing from the client to the server. If a second server does not start to listen to the IP of the failed server but always listens and serves on the same IP as another server, this is called Anycast (RFC1546) [6]. In the case of Anycast, any packet is delivered to the nearest known host (measured in router metrics like number of hops) which can be used for a simple load distribution. But this also means that if a server fails, some routers will update their tables to route packets to the nearest remaining server with the same IP. As described above, Anycast is already in use today.

The challenging part in the usage of Anycast with a connectionless protocol like UDP is the fact that a request (*F1*) could hit *Proxy 1* of the federation, but the response (*F3*) could hit *Proxy 2* (also a member of the federation and the Anycast group) as illustrated in Figure 4.3. This problem arises because for the routers behind both users

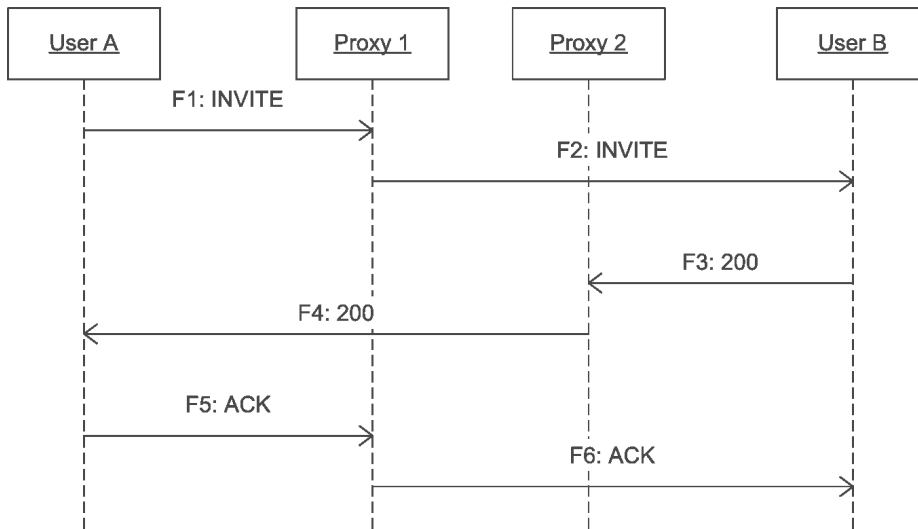


Figure 4.3: Split routing with Anycast and stateless proxies

the "nearest" server could be a completely different one. E.g., for the routers behind *User A* the "nearest" destination for the Anycast address is *Proxy 1*, but for the routers behind *User B* the "nearest" destination for the Anycast address is *Proxy 2*.

#### 4.1.1 Stateless Proxies

If the servers from the federation work statelessly then this split routing of the request and the response is no problem, because if the servers work correctly, request and response will reach the UA's as shown in Figure 4.3. And if one of the servers from the federation is offline, the UA's will retransmit the request or response until they receive a response. But the removal of an offline server from the Anycast group should be as fast as possible to prevent UA's from trying for too long to still use this instance of the Anycast proxy group. So a stateless proxy federation which uses and offers only UDP as a transport protocol is the perfect service for Anycast, as described on page 3 of RFC1546 [6].

The usage of TCP with Anycast is unfortunately not that straightforward. RFC1546 describes that a host which listens on an Anycast address should replace the Anycast address with the unicast address of the host in the returning SYN-ACK packet. Now the host which sends the SYN packet has to match the incoming SYN-ACK packet with the "wrong" source address only with the combination of source- and destination-port and the sequence number. With this solution the initiating host would learn the unicast address of the responding anycast host. Unfortunately, this solution requires specific and explicit support from the operating system of the initiating host. And additionally, we fear that this solution will not work with any firewall and NAT which is currently in use.

But as most of the implementations still use UDP for SIP transport we will investigate this further.

## 4.1.2 Stateful Proxies

The biggest problem appears when the proxies within the Anycast federation are stateful. (We assume UDP as the transport protocol from here.) Statefulness is often used because it is necessary for features like e.g., accounting, forward on busy or firewall traversing. In this case, it is a problem if the reply does not visit the proxy which delivered the request, because if the proxy does not see the reply, it will retransmit the request and finally send a 408 Request timed out back to the UA. Two solutions are possible for this problem:

- The proxy which forwards the request uses its unicast address in the Via and Record-Route header and as the source IP address. Unfortunately, using unicast address in requests eliminates redundancy for delivery of SIP replies.
- It uses the anycast address in the SIP headers as an address for all further packets. The problem is that replies to requests may visit a different server with the same anycast address and cause stateful processing to fail.

None of the solutions is perfect. An idea to solve the problem with the more attractive second solution is that the proxy inserts two Via headers into the requests which it forwards. The topmost Via header contains the Anycast address of the federation (as the IP or DNS name does not matter), and the second header contains the unicast address of the forwarding proxy.

If the reply with the double Via header hits the owner of the unicast address again, it simply removes the Via header with the unicast address and the second Via with the anycast header and delivers the reply upstream. What happens when the reply hits one of the other proxies from the Anycast federation is illustrated in Figure 4.4. The receiving *Proxy 2* detects that the reply is destined for the Anycast group, but at the next Via header it sees that the request was delivered by another group member. So it simply removes the Via header with the Anycast address and forwards the reply stateless to the unicast address of *Proxy 1*. Now *Proxy 1* can match the reply to the running transaction from the INVITE, removes its Via header with the unicast address, and delivers the reply upstream.

Unfortunately the same mechanism does not work for the Record-Route header. The Record-Route header is required if the proxy needs to be call stateful, which means the proxy is aware of all dialogs and not just transactions. This feature is required for use cases like prepaid calls or NAT and firewall traversal.

If *Proxy 1* first inserts its unicast address and then the anycast address into the Record-Route header then the UA of *User B* sees the header entries in the correct order and generates a Route header with the anycast address in front of the unicast address. But because the UA of *User A* reverses the entries of the Record-Route header from the received response the generated Route header would contain first the unicast address and then the anycast address. There are two possible solutions to guarantee that requests are delivered first to the anycast address from both directions.

- Include three entries into the Record-Route header which contain the anycast address at first, the unicast address as second and the anycast address again as third value.

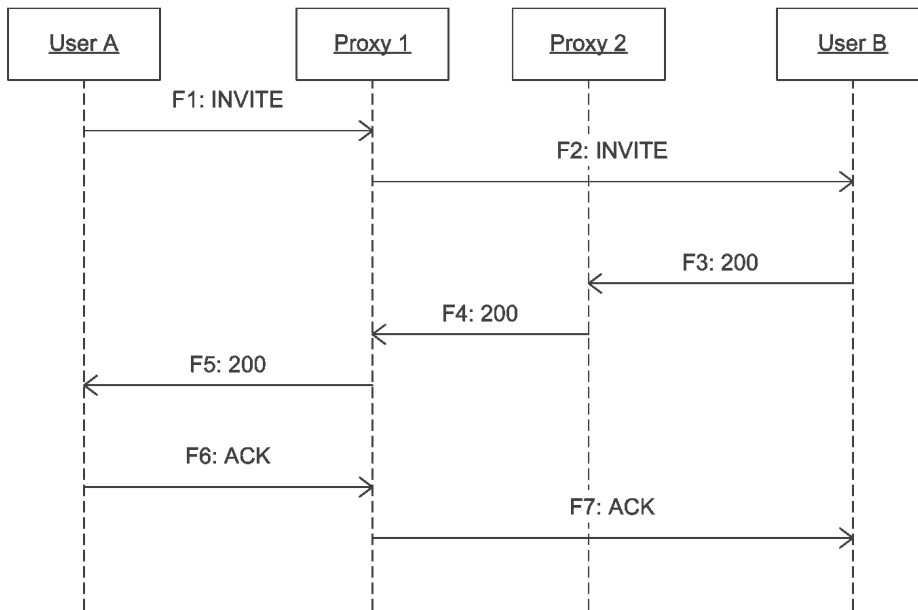


Figure 4.4: Non-straight signaling with the usage of Anycast

- Only include the anycast address in the Record-Route header and put the unicast address as additional parameter into the same header field. (Example can be found in the message at Appendix B.)

Because the first alternative adds a lot more processing overhead and enlarges the size of the requests but has no other advantages over the second alternative it is the inferior choice.

If one of the anycast proxies receives a request which contains a foreign unicast address as parameter of the anycast address in the Router header it should forward the request to this unicast address, if this proxy is online, instead of delivering the request.

A little more problematic is the case when one of the proxies of the Anycast federation fails during a running transaction, as presented in Figure 4.5. In this case the failed proxy can not relay the reply and acknowledgment for the request and the remaining proxies of the Anycast group have to be prepared to stand in for the failed proxy (and should obviously avoid to relay the reply and ACK to the failed proxy).

*User B* sends its reply to another member of the Anycast group, *Proxy 3*. This proxy removes the Via header with Anycast address but it has to know, or detect, that *Proxy 1* is offline. This is because in the optimal case it would not even try to deliver the request to *Proxy 1*, but would also remove the second Via header with the unicast address of *Proxy 1* and deliver the reply upstream to *User A*.

For negative replies *Proxy 3* could theoretically try to reconstruct the state of this transaction from the reply, to keep the stateful state. For 200 OK replies this is not necessary because the ACK does not belong to the transaction anyway. But Figure 4.5 also il-

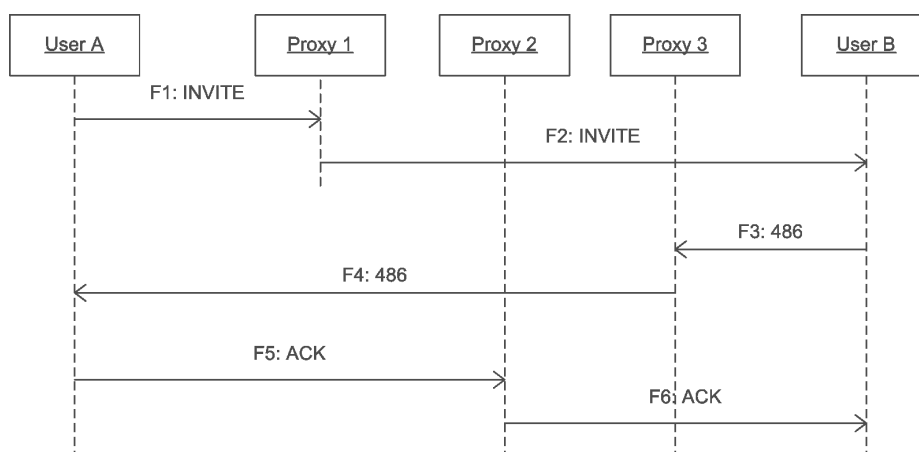


Figure 4.5: Anycast with a failed stateful proxy

illustrates that that does not make much sense because the ACK from *User A* can or probably will again hit another proxy from the Anycast group, *Proxy 2* in this case.

Also *Proxy 3* cannot include its unicast address in any of the normal headers in a reply, in order that *Proxy 2* recognize and forward it. And a new header would probably not be included by the UA of *User A* into the ACK to *Proxy 2*, so it can only forward the reply statelessly, as the ACK has to be delivered stateless to *User B*.

If *Proxy 1* fails after it delivers the reply upstream to *User A*, we can only hope that *Proxy 1* is removed fast enough so that the ACK is delivered to one of the other Anycast group members. And if a proxy fails after receiving a request or reply, but before forwarding it, we can also only hope that the failed proxy is removed from the Anycast group as fast as possible, so that a retransmission of the request or reply will be delivered to one of the remaining members of the Anycast group.

To resume with some changes the deployment of Anycast with IPv4 for SIP servers is possible. But the problems with TCP and the fact that the group membership management has to be really fast, but it will end up in host route updates which are normally not that fast, makes the usage of IPv4 Anycast doubtful.

## 4.2 Sharing User Location Database

The information where a user can be contacted at any time is kept in the user location database (ULD) and is part of the registrar in the SIP architecture. To reach the goal that a user is reachable all the time, independent of the state of its primary server, we have to duplicate the information as to where a user can be contacted.

The duplication of information is called replication. With the replication of the contact information the reachability of the user can be improved dramatically. User unavailability can then only occur if all of the federation servers become unreachable. Chapter 2 of [28] gives a good overview of the different approaches for replication.

Unlike replication for improving access times or speeding up parallel applications like in [5], we do not have the choice of replicating only to well-chosen peers or letting the peers fetch the information if they need it. This is because if we want to improve the availability we have to consider that any of the peers can fail at any time. Also, the alternative of invalidating the copies at the other peers versus updating the complete copy is not a choice for high availability scenarios, because invalid copies cannot be used anymore for locating or contacting the user. And besides this, a typical contact update is not so big that an invalidation message would be advantageous in terms of network load or processing speed.

#### 4.2.1 Client- vs. Server-based Replication

The contact information of the user could either be replicated by the UA itself or by the server, in this case the registrar.

If the replication is done by the UA itself, it would be done with multiple registrations at all the servers of the federation. The advantage is that this is done over SIP and requires no additional protocols.

Replication by the server does not have to use the SIP protocol. In this case the replication could be done over any protocol of the persistent storage of the server. In most cases this would be the replication feature of the database management system which the server uses for persistency, but a replication over filesystems or on storage management level is also imaginable. However in all these cases the servers of the federation have to use the same replication mechanism, which means they have to use the same database, the same filesystem, or whatever. So the most flexible alternative for the replication on the server site is the SIP protocol, because all the peers of the federation support this protocol. This makes the peers of a federation independent from each other in terms of the used registrar implementation, the database, any other system parameters and allows the usage of memory caches for the ULD.

Very often a two phase commit (2PC) protocol is used in distributed applications to ensure a synchronized state over all participating nodes [5, 28, 41]. In a 2PC protocol the receiving server first sends the request to the replication destinations and waits for their acknowledgments. After it received all acknowledgments from the replication destinations it sends out the reply to the client. But this practice is not usable in our scenario. Firstly it would introduce very long delays in the response time of the registrar, which is not desirable for signaling traffic. Secondly, a 2PC protocol introduces the possibility of blocking, as described in [41]. The potential blocking problem of a 2PC protocol can be solved by a three phase commit (3PC) protocol. But the delay in a 3PC protocol is even bigger than in a 2PC protocol, and is therefore no option for the real time requirements of the SIP protocol.

Thirdly, for a non-SIP-based implementation it would be difficult or maybe impossible to receive an acknowledgment from the replicating layer of the SIP server that the replication was successful. For example if the database do not secure the replication with distributed transactions it is not guaranteed for the SIP server that the replication succeeded. And lastly, we have to consider that a peer could fail, which would impede a first commit and finally result in an error response to the UA. This means multiple phase commit implementations are not possible, and we have to acknowledge the change first

to the UA and then propagate the change to the other peers. However, this introduces the small chance that the state change is not synchronized to the other peers because the replicating registrar failed before it was able to propagate the state change. In general, an optimistic version of replication that is failure tolerant and that achieves a better response time should be better for SIP.

The consistency problem of distributed objects does not exist for contacts, because only one entity, the Address of Record (AOR) owner, changes the contact. And we do not have to consider operations like updates or the increase or decrease of values from the contact, because contacts are only read for lookups of users and overwritten if the owner of the contact sends a new REGISTER. The synchronization and serializability of the changes to a contact are not a problem because of the increasing CSeq number within every change. In [41] this design is called FIFO consistency, but we have additionally the serializability through the CSeq numbers, which is not a requirement for FIFO consistency.

UA-based replication requires first that the UA knows all peers of the federation. And secondly, all the peers of the federation have to be able to authenticate the UA over his password. The advantage of this alternative is that it is relatively transparent for the servers because the UA just registers normally over SIP at all the registrars. The disadvantage is that the peers have to share the passwords for all their users (see Section 4.3 for details), and that it requires changes at the client side. This is because many UA's are able to register at multiple destinations at the same time, but we do not know any client which is able to do this with the same account for one domain at different servers at the same time.

Replication by the registrar is the better alternative. In this case the replication is completely transparent for the client and requires no changes at the UA, because the UA just registers at its server as usual and the rest is done by the server. It requires some changes at the server, which replicates an incoming contact to one or more of its peers. But it has the advantage that the trust relationship between the client and the server can be transferred to the other peers, with another mechanism that does not need the password of the client to be able to verify the authentication header of the client.

## 4.2.2 Trust Relationship

As mentioned above, the authentication of the client and the resulting trust relationship between server and client relies on a shared secret, the password. With this password, only the client is able to answer the challenges from the server<sup>1</sup> and get access to change its own contact information in the user location database of the registrar. The UA-based replication can only rely on the password or the H(A1) known by all peers and the client. Since the peers of the federation may be not trustworthy enough to share the passwords of the users with them another authentication solutions should be used. (See Section 4.3 for details.)

With server-based replication, the trust relationship can be transferred with a share secret between two peers or among the whole federation if the SIP protocol is used for replication. Then the replicating server would use the shared secret to answer challenges from the other peers, or it would replace the authentication header within the REGISTER

---

<sup>1</sup>SIP uses an adapted version of Digest Access Authentication from RFC2617

with its own authentication header. Answering challenges would add an additional load to both servers and replacing the header in advance with the correct nonce and response would require that both server use identical nonce creation algorithms. But the trust relationship can also be transferred over a secured transport layer. This can either be a TLS connection over TCP between the peers or an IPSec connection. In this case the peers would simply trust the incoming REGISTER from the other peers based on the source IP in the case of IPSec or the TLS credentials.

### 4.2.3 Network and Processor Load

From the perspective of network load there is no real winner between the two alternatives. In the case of UA-based replication, the network load on the Internet connection of the UA would multiply with the number of peers in the federation. For the server-based replication, the network load multiplies with the number of the peers in the federation too, but at the Internet connection of the primary server. From the view of the backup peers there is no big difference between the two alternatives, because in both cases they receive the additional traffic, only from different sources. Only in the case of server-based replication without using SIP could the network load for the backup peers be lower, because other protocols could probably transfer the same information in less bytes than an ASCII-based protocol like SIP.

If we look at replication from the view of the additionally required processing power, again none of the solutions is better. Merely sending the same request to a different destination would not require much additional processing power for the UA. We should consider, that the UA may have only a very limited processing power, and that mobile devices have to save battery power as much as they can. If the UA has to answer additional challenges from the backup peers, this could add, depending on the number of peers in the federation, a lot more processor load to the UA. On the other side the average processor load on an UA should normally be very low. For server-based replication, the amount of load added to the processor depends on the implementation of the replication. If the server just sends a copy of the request to the other peers this would not add any significant load to the processor of the server. If the server has to answer challenges or prepare the requests in some way for replication however, this would add additional load to the server. In the case of non-SIP server-based replication it is very hard to say how much load this would add to the server.

### 4.2.4 Failed Replication

Another point of interest is how failed replication impinges on the replicating unit. It is obvious that all peers of the federation should have the same entries in their user location databases, and they should be synchronized as much as possible. Thus if one of the peers loses its synchronization in the replication, either because of a planned maintenance or of a failure, one of the peers has to take care that the failed peer gets synchronized after its recovery, or the failed node has to pull for updates after its recovery. In non-SIP replication, it depends on the implementation as to whether a pull or push of the updates will keep failed peers up to date. In the case of UA-based replication, the UA itself has to keep retrying the replication until it succeeds, because the failed peer does not know where it should fetch updates after its recovery, except to

pull for the updates from the other peers. But this would break the transparency of the server in the UA-based solution. Furthermore, the UA has a problem replicating its own unregister<sup>2</sup> because we cannot assume that the UA is online any longer after its own unregister. Unregisters should be replicated too to remove the contacts of the user from the ULD's off all federation members (to prevent unnecessary connection tries to this contact). If the failed peer polled the update in the UA-based scenario from other peers it could also fetch the information that a contact is unregistered from the other peers, but then all peers have to store all unregistered contacts only for the case that one of the other peers needs to fetch this information. Besides all these problems we should also consider the probably limited capabilities of the UA, and that replication retries consume additional memory, processor power and network capacity. From all the facts about UA-based replication it is clear that this is not really an alternative to implement. For the server-based SIP replication scenario it does not do any harm that a state has to be kept for replication because this state has to be present in the registrar anyway. The implementation of server-based replication needs to change slightly: unregistered contacts are not removed until replication to all peers successfully completes. (See the next paragraphs for the details of the required changes.) The advantage of server-based replication is that all the unsuccessful replicated contacts for one failed peer can be stored together and then submitted as a large group after the recovery of the failed peer. The additional processing power and used network capacity for retrying to replicate should not harm a server. Only the additional memory or storage needed for storing the unregistered contacts could be problematic.

#### 4.2.5 Cache Updates

If caches are used for the ULD, like in the case of the SIP Express Router, it becomes a problem if the replication happens at a service layer below SIP. Then each update at the replication layer have to update the content of the cache too. Either by updating the content of the cache or by removing the old invalid cache content. The worse alternative, in terms of performance, is to turn off the cache completely and use only the database for look-ups. But such implementations are currently not yet available for the SIP Express Router. To avoid the reimplementation of the cache, we decided to use the most transparent method of replication, using SIP itself. With the use of SIP for replication, it does not matter how each of the peers stores the user location database, and even plain text files or memory-only storage is possible.

#### 4.2.6 Peer Status

If the SIP protocol is used for replication the detection of the failure of another peer appears problematic. The status of the other peers should be known to know which peers still participate in the replication and which peers need an update after their recovery. There are two choices as to how the detection can be carried out, peer status checks or per instance checks. With the peer status checks you always know the status of a peer because you store the result of a periodically test. In contrast with per instance checks you do not know the status of a peer all the time, because you only store if this

---

<sup>2</sup>Unregister means the removal of a contact by sending a REGISTER with expires set to zero.

particular replicated unit was acknowledged by the peer and delete all informations on success.

The first one, peer status checks, tests the status of a partner periodically with pings is well-known from classic HA scenarios. The difference between a permanent connection<sup>3</sup> and a periodic test without a permanent connection does not count, because in both cases the sending side stores the status of the receiving side. Normally the sending side will also check against this stored status before it sends data. The only difference between a permanent link and periodic tests is how up-to-date the stored status is.

The second alternative, per instance checks, relies on the replies from the peer. But we are not interested in knowing the state of the peer all the time, because we do not have to take action if the other peer fails, as in classic IP takeover scenarios. And we should not produce additional network and processor load on both sides of the replication with an additional state checking command or request if no changes happen in the user location database. Thus we rely on the fact that the transaction management will keep track of the success or failure of the replicated REGISTER. Only if a peer fails do we want to know when it will recover. But we have to be sure that the registrar of the peer is working again. Successful ICMP echo requests or SIP OPTIONS requests do not guarantee that the registrar is working. Because of this, we decided to resend the first failed REGISTER at regular intervals to check if the failed peer had already recovered. If the peer did not acknowledge a REGISTER, the transaction would be stored for a second replication attempt in the future. If the peer acknowledged a REGISTER successfully, this could be interpreted as the recovery of the failed peer.

#### 4.2.7 Push vs. Pull

An alternative to pushing the updates to a failed peer after its recovery could be for the failed peer to pull the updates from the other peers. This would have the advantage of minimizing the delay between the recovery and the point in time where the peer is once again synchronized with all other peers. The most simple solution is that the recovered server fetches the updates from the running server. But as it does not know what has changed during its outage it either would have to fetch the complete ULD or the running server would have to store all updates in a special area from where the recovered peer pulls all stored informations until its up-to-date again.

The other pull solution would require that first, a failed peer could announce its recovery, and secondly, that the replicating peer would have to listen for such an announcement. This announcement and listening process could either happen over a new protocol or service, or over a new or specially prepared SIP request, although this would require bigger changes on both sides of the replication than the push alternative does. The disadvantage of the delay between recovery and being synchronized again could be minimized if the interval between the resending of retries is configured accordingly.

#### 4.2.8 Keeping Removed Contacts

To prevent contacts from immediately being removed from an unregister, we introduced a new state for them, which we call the zombie state. We need this to be able

---

<sup>3</sup>This connection is only virtually permanent, because also over this connection periodic tests have to be sent. If no data is sent, for example, over a TCP connection the failure of the other side would not be detected.

	UA-based SIP	Server-based SIP	Server-based non-SIP
trust relationship	shared password or token authentication	shared password or TLS	protocol-based or TLS
network load	n-times (n=number of peers) multiplied on the UA uplink	n-times multiplied on the server uplink	up to n-times multiplied on the server uplink
processor load	up to n-times on the (small) UA processor	up to n-times on the server processor	up to n-times on the server processor
retries	expensive and might be impossible	little additional memory and processor load	little additional memory and processor load
cache synchronization	no problem	no problem	problematic
state watching	SIP-based	SIP-based	implementation dependent
push or pull	only push possible	push requires no protocol changes	push and pull possible

Table 4.1: Replication alternatives comparison

to replicate an unregister request later if the peer is currently not available but its user location database contains the original contact from an earlier REGISTER, otherwise a failed peer would forward requests to the no longer valid contact address after its own recovery. This does not really do any harm because the request then should normally time out and would be answered with 408 Request timed out. But this would introduce longer delays in call establishing and result in more inconvenience for the users. This means that on an unregister request we do not remove the contact from the user location database, but we turn it into a zombie. Zombies are only removed from ULD if they are no longer needed for replication.

#### 4.2.9 Delta vs. Full Transfer

One of the ideas behind server and SIP-based replication is that we only transfer the delta<sup>4</sup> of the changes from our ULD. As a result, normally expired contacts are not turned into zombies but removed from ULD, because these contacts would also have expired at the other peers and so we would not have to replicate them. Likewise, expired zombies are no longer kept in ULD and no longer replicated because they would also have expired at the replication destination. But this delta transfer requires that a peer use a persistent storage for its user location database from where it can read in the ULD to get a state similar to that before the outage. In this scenario every peer is responsible for replicating the changes from its ULD to the other peers, where the change was received from a UA and not from one of the other peers. This has the advantage that

<sup>4</sup>Delta means the mathematic  $\Delta$  as the difference of two sets.

no master for the replication has to be elected, which would introduce a new single point of failure (SPoF). The replication master as SPoF is often prevented with a state-watching function in every peer which observes all other peers. On the detected failure of the master replicator a new master is elected from the peers that are still alive [7]. We want to prevent the introduction of the state-watching function and the election algorithm, although the non-use of a replication master introduces the possibility of an inconsistency in the case of overlapping outages of multiple peers.

An alternative to delta transfer could be the transfer of the whole user location database to a recovered peer. But this produces a large network and processor load on both sides that is dependent on the number of contacts in the ULD. Particularly for small outages, this solution is also a big overkill just to transmit the changes during the outage. So this solution should only be chosen if the peer has no persistent storage available for its ULD. In this case, a recovered peer would restart with a totally empty user location database, and the transfer of a complete ULD from one of the other peers would be the best and only choice to get back in synchronization as fast as possible. Problematic in this scenario is that one of the peers has to be elected to prevent all peers from transferring their ULD's to the recovered peer.

It is also desirable that contacts which are registered and unregistered during the failure of the peer are removed from replication. Depending on the implementation of the registrar, the replication of register and unregister may also simply not be possible, like in the case of the SIP Express Router (see Section 5.1 for details).

As an example the call flow from Figure 4.1 with enabled SIP replication now looks like in Figure 4.7.

#### 4.2.10 Persistency and Synchronization

Making the replication information persistent may be desirable to be able to continue the replication after a restart of the server. This should be no problem if all the UAC's store and increase the value of their CallSequence (CSeq) header, as described in section 10.2 of RFC3261 [36]. Currently only a very few UAC's are fully RFC3261 compliant, and we are aware of many UAC's which reset the value of their CSeq header to the starting value during every restart. But with these unincreased CSeq values the peers of a federation cannot safely distinguish between old and new REGISTERS. So the feature of restarting the replication with the replication information imported from a persistent memory on server startup should be made optional until all UAC's within a federation are fully compliant to RFC3261.

Our replication design also has a disadvantage in that the synchronization between the peers of the federation is not one hundred per cent safe. For example a UA can disturb the synchronization if it changes its contact at two servers at nearly the same time but with different values. This would result in a race condition between the replication messages of both servers to each other and to the other peers, and a synchronized state could not be assumed. But we are not aware of a UA which is able to register at different servers at the same time for the same account. And above all, the UA would just risk its own reachability with such behavior and we assume that a UA is interested in being reachable.

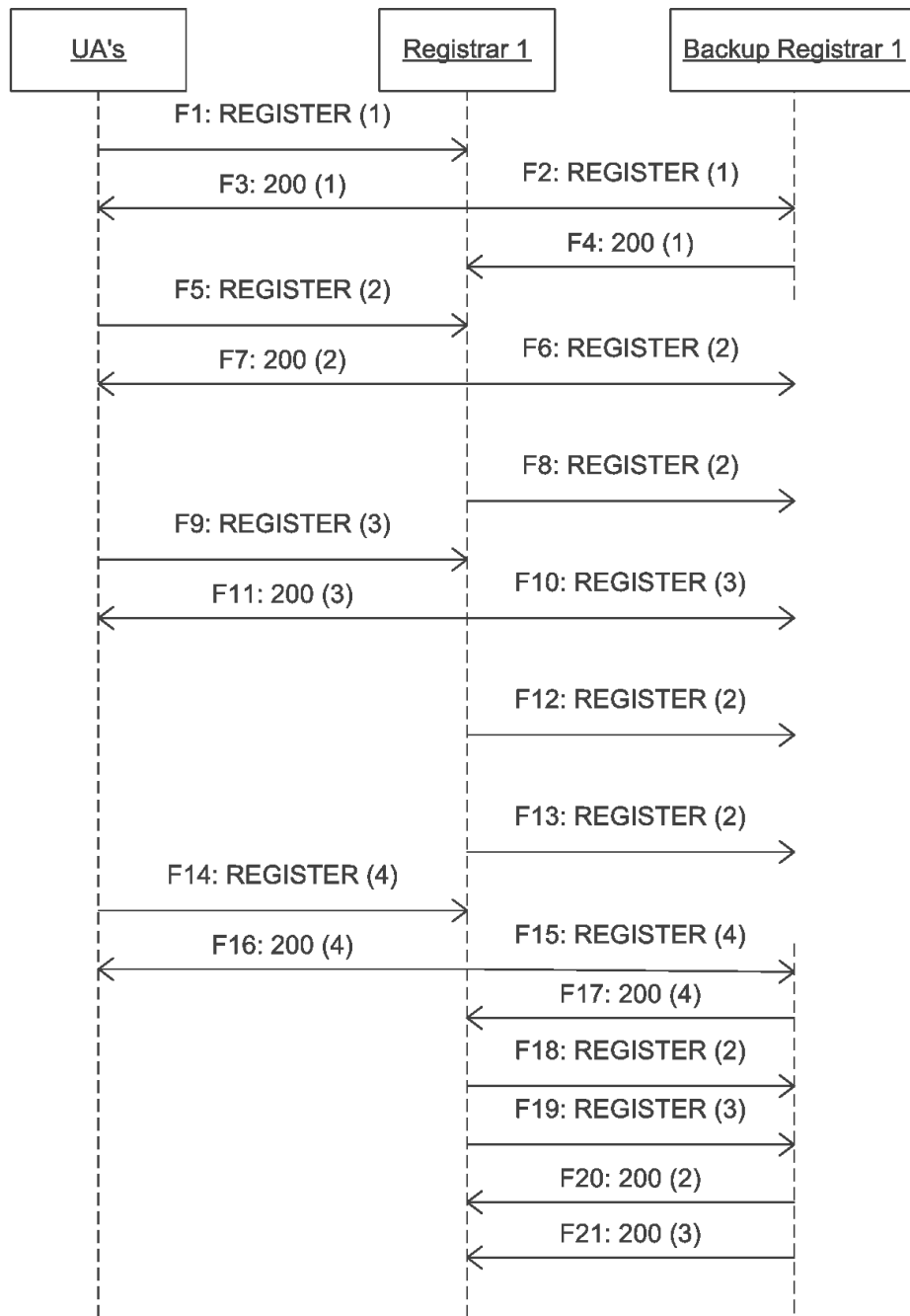


Figure 4.6: Example SIP replication with backup failure and recovery

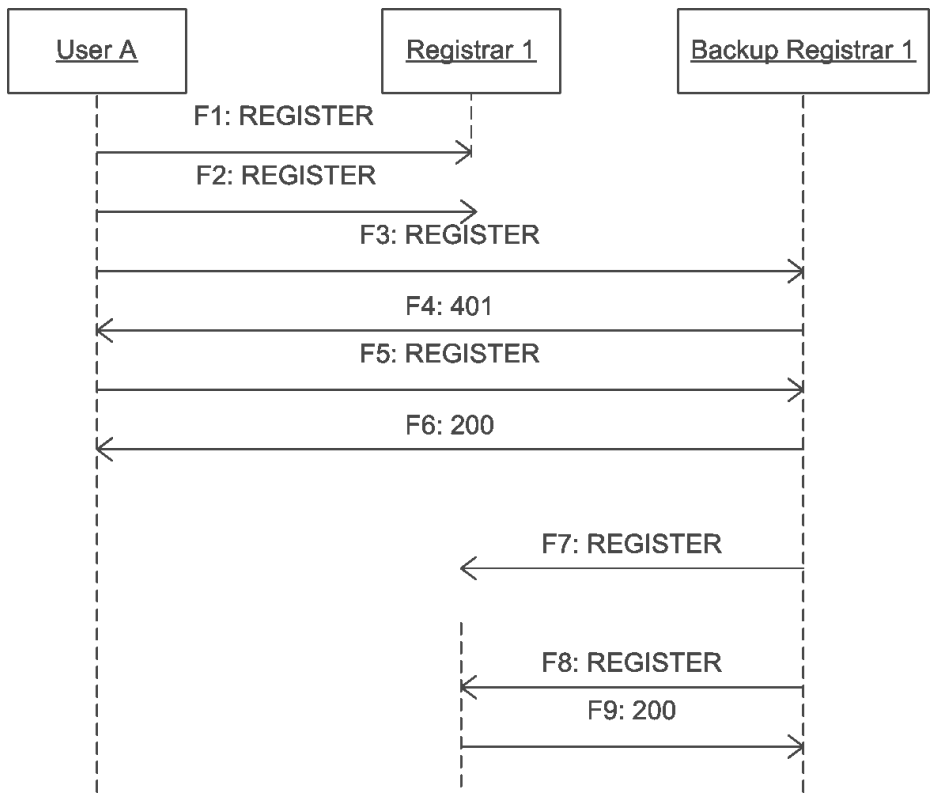


Figure 4.7: Registration at a backup with SIP replication

### 4.2.11 Multicast

As described in chapter 7 of [41], Multicast is a good way to realize a reliable group communication. Especially if the feedback is suppressed, it can save a lot of bandwidth in a group communication like replication to multiple peers. Although for Multicast only the unreliable UDP can be used for transport, so that the application has to care for reliability.

But for several reasons we did not consider Multicast in our ULD replication design. First of all, the use of Multicast in the Internet is still problematic because it cannot be guaranteed that all routers between the members of a Multicast group will be supporting Multicast. Secondly, the core of SER would need to support the multiple responses for one request, to distinguish which peers acknowledged the request and which not. The other solution is to suppress the response and to send a special response only if the fact that a peer recognized that it missed one of the previous messages creates some trouble for the SIP replication. This is because any new message overwrites the values of previous messages for this contact, so only a new header field which makes all requests from one host serializable could enable a peer to recognize that it missed one of the previous messages from this peer. As described in [41], this requires that all messages from a host be stored for some time, to be able to resend them again. Above all, this setup requires that a special group management is established for each Multicast group, because every sending host has to know which peers should receive its requests before sending them, and which peers have left the group before it started to send.

### 4.2.12 Conclusion

SIP-based replication is the most flexible and transparent solution to keep a federation of SIP registrars synchronized. The easiest solution is to simply forward the incoming REGISTER request to the backups. But as failed backups should be updated as soon as possible after their recovery and the optimal replication will only transfer the last state change of a contact to failed peers, and only if this state change is needed by the peer, the optimal SIP-replication requires some small changes to the registrar implementation.

## 4.3 Sharing Credentials

No ultimate solution exists as to how a backup server can authenticate the users of one of the backup domains without knowing their passwords, so we present here several solutions with different dis-/advantages. The best choice depends on the detailed requirements of each deployment scenario.

Of course, you can replicate your subscriber database to all servers from the federation, but this causes the problem of database interoperability and you have to trust the operators of the backup servers absolutely. We assume that the backup servers are trustworthy enough to provide an emergency fallback service for you, but you would not consider giving the passwords of all the users to them.

A prerequisite for all the methods discussed below is that the UAC is capable of using multiple SRV records, as explained in Section 3.3. Each solution which we discuss will

Implementation effort	Very low	Trustworthiness	Not required
Management effort	Very low	Former member	No problem
Client effort	Zero	Traffic between servers	Low
Compatible	Yes	Primary contact	Required

Table 4.2: Trusted location

end with a table which gives a short overview of the dis-/advantages of the solution. We now explain the fields of these tables:

**Implementation effort:** how difficult the solution is to implement. This includes how many problems should be expected in the deployment of this solution.

**Management effort:** how large is the effort to manage the passwords, or whatever is used instead of the passwords, for this solution?

**Client effort:** does the client have to undertake additional procedures or need more processor power to support this solution?

**Compatible:** does this solution work with the existing UAC on the market?

**Trustworthiness:** does the primary server, or more precisely the operator of the server, have to trust the security of the backup servers and its operators? The information at the backup servers can be misused to authenticate at one of the other servers with a faked identity.

**Former member:** are former members which are not trustworthy anymore a problem? They could misuse old information.

**Traffic between servers:** how high is the traffic between the servers in addition to the traffic which is required for the sharing of the user location database?

**Primary contact:** does the UAC have to contact its primary server before it can use one of the backup servers? For example, if the UAC is turned on or started during the outage of its primary server, can it directly use one of the backup servers or does it have to wait for the recovery of its primary server?

### 4.3.1 Trust the Location of the Registered Contact

A very simple solution is to trust the location of the registered contact. This means the backup server would simply allow any request from an IP which matches the IP of one of the registered contacts for this user.

But this will only work if the user was able to register itself at its primary server. This means that a user cannot register if it turns its client on during the failure of its primary server. Also, moving from one location to another would also be difficult, because the user has to register the new location first from the old location before it can move to his new location. However, this solution is completely insecure, because nearly everyone on the Internet can spoof the source address in their IP packets.

Implementation effort	Low	Trustworthiness	Required
Management effort	Low	Former member	Problematic
Client effort	Zero	Traffic between servers	Medium
Compatible	Yes	Primary contact	Not required

Table 4.3: Replicate H(A1)

### 4.3.2 Replicate H(A1)

Another solution if you do not want to share the passwords of the users with all the peers is to share the hash of the A1 string (H(A1)) from the DigestAuth (for details about H(A1) see Section 2.3.5). The advantage is that it works with all existing clients and the backup peers can also simply use the normal DigestAuth scheme. Only the calculation of the H(A1) string does not apply for the backup peers.

The replication of this H(A1) string can either be done by database replication or with the SIP protocol. In both cases, the connections between the servers should be encrypted to prevent the sniffing of the H(A1) string by an eavesdropper. Replication over databases requires no changes at the SIP servers, but it requires the same databases at all servers, or at least replication between the databases has to be interoperable. Thus we prefer to replicate H(A1) over the SIP protocol. A new header field `X-User-Authentication` within a Notify message could carry the username and the hash of the A1 string. To be safe for the future, the hash algorithm which was used to calculate H(A1) should be included in this header field too. It is obvious that a method is also needed to remove a user and its H(A1) from all the backup peer databases, which is necessary when a user is removed from its primary server for whatever reason. Again, a Notify with a `X-User-Authentication` header but with an empty H(A1) could indicate that the backup peer has to remove this user from its authentication database.

But this solution has two big handicaps. Firstly, you have to trust all the backup peers, because everyone who has access to the H(A1) strings at the backup servers can use them to fake their identity at all servers of the federation. Also, if it does not know the password of the user it can answer any DigestAuth challenge of the primary or the backup servers. Thus he is able to register and to place calls at any of the servers. Secondly, former federation members which are not trustful anymore still have all the H(A1) strings of all the users from a federation. The only way to prevent former federation members from misusing all the H(A1) strings is for all users of the federation to change their passwords. This is a big inconvenience for all the users and a big management job (because of all the new synchronization) if this case should ever happen.

### 4.3.3 Daily Pass for all Servers

The next alternative is an adaptation of short-lived cookies from the web which we call Daily Pass. In short this means we handover a special token, the Daily Pass, after a successful registration to the UA. On a failure of the primary server, the UA uses this Daily Pass to calculate the response to the challenge of the backup server. Also, the backup server can create the Daily Pass on its own and check if the response is correct for this Daily Pass.

After the user is successfully authenticated at its primary server, normally over DigestAuth, we have to send the Daily Pass to the user or to the UA. So the best way to send the Daily Pass to the UA is to place it in the 200 OK response to a REGISTER request. Because there is no Authentication header in the 200 OK response, we need a new header field for this purpose, for example `Daily-Pass`. To prevent the sniffing of this Daily Pass by an eavesdropper, we need to encrypt the content of this header field. The user and its primary server both know the password of the user as a shared secret, so this can easily be used for the encryption of the content of the `Daily-Pass` header. To be compatible in the future, the encryption algorithm used should be included into the header in clear text.

As the Daily Pass should be time limited, it has to contain an expires value. Also, because the Daily Pass token itself is only a hash value, we also have to pass any information in clear text that is needed by a backup server to recalculate the Daily Pass hash value. We cannot assume that the clocks of all participants are synchronized. So we have to give the date and time when this Daily Pass will expire as an absolute SIP-date as in the Date header, according to RFC1123 [19]. If we gave the expires value as an integer with seconds relative to a given date, the client would have to recalculate this value every time it tried to authenticate itself. But if the recalculated date and time values did not match the original values used during the creation of the Daily Pass, then authentication would fail because the backup server would be able to create the same Daily Pass token as was sent to the UA by the primary server. To reduce the parsing effort the usage of an integer as expires value, which contains the seconds since 1st January 1970, should be preferred.

Finally, we should also include the hash algorithm which was used for the creation of the Daily Pass token to be able to change the hash algorithm in the future.

An example of a `Daily-Pass` header from a 200 OK could look like this:

```
Daily-Pass: dp="alb467...f41283c9", encrypt-algorithm=AES,
expires="1057792838", hash-algorithm=HMAC-MD5
```

If a UA now tries to authenticate itself at one of the backup servers it will use the decrypted value of the `dp` field from the `Daily-Pass` header instead of its password to calculate the H(A1) for the DigestAuth. To signal that it is not the password that is used to calculate response but the Daily Pass, the token "Digest" within an Authorization or Proxy-Authorization header has to be replaced with "Daily-Pass-Digest".

On the one hand, the user and the backup have no shared secret to encrypt any header field. Thus we will not include the Daily Pass in any request from the UA to a backup server. On the other hand, we want to prevent the establishment of a secured connection between the UA and the backup server, because of its overhead and delay.

This means we have to pass all information to the backup server which it needs to recalculate the Daily Pass. In detail that means the expires value and the hash algorithm used in the `auth-param` token of the BNF of the (Proxy-) Authorization header according to the RFC3261.

An example `Authorization` header which uses the Daily Pass could look like this:

```
Authorization: Daily-Pass-Digest username="foo", realm="bar.com",
nonce="dcb645...23f6c1", uri="sip:foo@bar.com", qop=auth, nc=1,
```

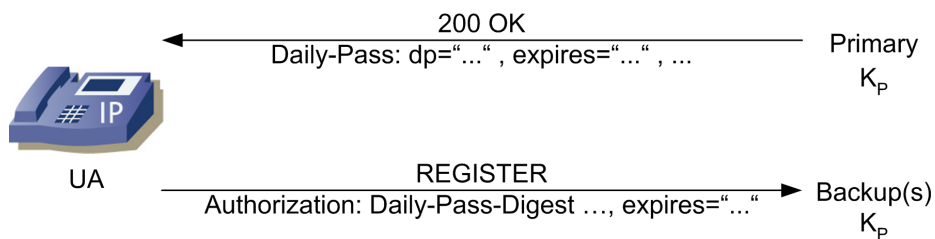


Figure 4.8: UA receives Daily-Pass from primary and uses it at a backup

```

cnonce="4512a4bc", response="ab3178...1104ca", expires="1057792838",
hash-algorithm=HMAC-MD5

```

Now the backup server has to recalculate the Daily Pass to check if the UA answered with the correct response. Figure 4.8 illustrates that the UA receives the Daily-Pass in a new header field within the 200 OK from its primary server and then uses it to authenticated its REGISTER (or any other request) at one of the backup servers.

First of all, the Daily Pass contains a shared secret which only the servers of the federation know, so that only the servers can create a daily pass. Secondly, only the owner of the Daily Pass should be able to use it, so we include the Address of Record (AOR) into the Daily Pass. And finally the Daily Pass should only be valid for an limited amount of time, therefore we include the date and time of the expires value into the Daily Pass. The resulting input string for the Daily Pass looks like this:

```

Daily Pass string =  $K_{P,B}$ , AOR, exp
 $K_{P,B}$  = shared secret key of primary server P and backup server B
AOR = Address of Record from the user
exp = Expires value of the Daily Pass as SIP-date

```

As we do not need or want to pass the content of the Daily Pass in the full reconstructable form, for example as encrypted text, we will just create a hash of the above string as a Message Authentication Code (MAC). Since  $H(K, M)$ <sup>5</sup> is known to be unsafe according to page 458 of [38], we propose to use the HMAC function from RFC2104 [30] with MD5 as the hash function to create the MAC as Daily Pass. This results in the following function to create the Daily Pass:

```

 $H(K_{P,B} XOR opad, H(K_{P,B} XOR ipad, AOR, exp))$ 
opad = the byte 0x5C as fill up for the block length of the hash function
ipad = the byte 0x36 as fill up for the block length of the hash function
XOR = bitwise exclusive-OR

```

<sup>5</sup>M = Message, in this case the concatenation of AOR and exp.

Implementation effort	Medium	Trustworthiness	Required
Management effort	Low	Former member	Problematic
Client effort	Medium	Traffic between servers	Low
Compatible	No	Primary contact	Required

Table 4.4: One Daily Pass for all servers

If one of the backup servers receives a request with `Daily-Pass-Digest` in the (Proxy-) Authorization header, it first creates the Daily Pass for the user as described above. Then it uses the result as a replacement for the password of the user during the creation of H(A1) to check the response of the DigestAuth.

To complete the function for the primary server to create the `dp` string within the `Daily-Pass` header:

$$dp = hex(E_{K_{U,P}}(H(K_{P,B} XOR opad, H(K_{P,B} XOR ipad, AOR, exp))))$$

*hex* = ASCII to hexadecimal conversion as described in section 3.1.3 from [18]

$K_{U,P}$  = shared secret key between user U and primary server P = password of the user

The advantage of this solution is that password synchronization between the peers of a federation is not required, and this results in a low management effort for passwords, keys or anything like that. But unfortunately, this solution is not compatible with existing UA's and it requires that you can trust the peers within the federation, because every server which knows the shared secret key of the federation can create a Daily Pass for every user from the federation. So a user or administrator with access to the key can take over the identity of any user from the federation. Also, former members of the federation are a security risk because they usually still know the shared secret key. If the shared secret key is replaced after the retirement or exclusion of any server, there is still a time window up to the time where the last granted Daily Pass expires, for all Daily Passes created with the old shared key, which become invalid immediately. That each UA has to authenticate at its primary server first before it receives a Daily Pass which allows authentication at one of the backup servers is also not very positive. And if all Daily Passes expire until the primary server recovers from a failure, then all the UA's must wait for the recovery of their primary server.

#### 4.3.4 Daily Pass for each Backup Server

To eliminate two disadvantages of the Daily Pass solution with one shared key for the whole federation, each server pair could use its own key. Figure 4.9 illustrates in the left graphic a) that it is not sufficient to use one key for each server of the federation, because the former federation member C would still know the key from A ( $K_A$ ) and B ( $K_B$ ) after its retirement or exclusion. To safely exclude a former member of the federation from faking Daily Passes is only possible if keys for every connection between each server pair are used as illustrated in Figure 4.9 in the right graphic b). This is because in this

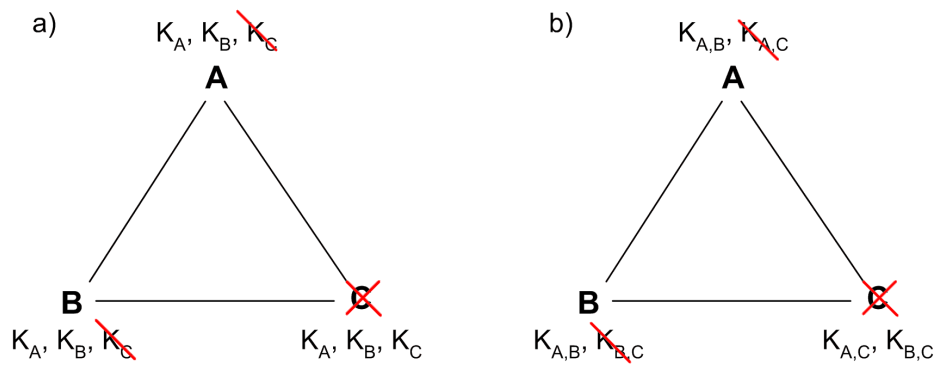


Figure 4.9: Difference between keys for each server and for each connection

case, server  $C$  only knows keys ( $K_{A,C}$  and  $K_{B,C}$ ), which are not accepted anymore by the remaining servers  $A$  and  $B$ . Note this do not change the number of Daily Passes which a client receives from its primary server, because it will not receive Daily Passes which are created with the keys for the connections between the backups.

But this also shows that key management is not easy for this solution. Each new server has to negotiate a secret key with each server of the federation, and on a retirement or exclusion of one of the servers, all the remaining servers have to remove or disable the key for this server immediately.

We have to create a list of Daily Passes for every client, because the UA needs a different Daily Pass for each server within the federation. The processing effort for the creation of all these Daily Passes increases with each additional server in the federation. Then we have to transfer a list of encrypted Daily Passes to the UA, although the UA still has to be able to differentiate all the Daily Passes, because it has to choose the correct Pass out of the list for the backup server at which it currently is trying to authenticate.

So this solution increases the management effort at the servers and requires more processing time at the primary server and the clients. However, untrustworthy former federation members are not a problem anymore. The UA could only try to use a backup server which has recently left the federation but for which the UA still has an unexpired Daily Pass. In the normal case, the former federation member would reject the request from this UA. In the worst case, the former member could pretend to still provide the normal backup service, but route the outgoing calls from the UA's of the federation to the wrong destination. But the user will still be reachable through the remaining members of the federation, and the former member cannot change this if the key for this server is removed early enough.

Another drawback of this solution, besides the extra processing power at the primary server and the UA's, is that the UA needs to contact its primary server first to receive the Daily Passes, so if a UA is turned on during the failure of its primary server it is still lost. Also, the whole solution is not compatible with the existing UA's.

We also considered using a Needham-Schroeder protocol like Kerberos to grant a ticket (see, for example, [38] for details about Needham-Schroeder and Kerberos), the Daily Pass in our case, to each of the backup servers, but the ticket granting server in this

Implementation effort	Medium	Trustfulness	Not required
Management effort	Medium	Former member	No problem
Client effort	Medium	Traffic between servers	Low
Compatible	No	Primary contact	Required

Table 4.5: Daily Passes for each server

solution requires the absolute trust from all federation members because it stores the passwords of all users, and this ticket granting server is a SPoF itself on which all servers in the federation depend. The trust problem may be solved for the cooperation of bigger service providers with a central trustworthy agency, but this seems to be out of reach for small federations of universities or for the example about the Fraunhofer Gesellschaft from the beginning. The SPoF problem could only be solved if the ticket granting service itself was highly available and optimally geographically distributed, otherwise the whole federation would fail if the authentication service is not running or not reachable.

#### 4.3.5 Distribute Precalculated Nonce-Response Pairs

The next approach is well-known from GSM networks. If a subscriber needs to be authenticated in a visited network, this network asks the home network of the subscriber for some challenges with precalculated responses. With this solution, the visited network can authenticate the subscriber without knowing the secret "key" of the subscriber.

In contrast to GSM networks, we cannot assume that we can ask the primary server for some nonce's with precalculated responses if a backup server needs to authenticate a user, because the primary server could be unavailable. So the primary server has to distribute or push the precalculated responses to prepare the backup servers for the case of a failure.

The pros of this solution are that it is compatible with the existing UA's, and it does not require that a UA contact its primary server first before it can use one of the backup servers. The problem of trusting the backups and former members of the federation does not exist if all backups get different precalculated responses. In this case, a backup server cannot use its knowledge to cheat another backup server, and also the precalculated responses finally transmitted to a now excluded member are not usefull anymore.

But this solution also has some problems. Either a nonce has a built in expires time, in which case the nonce-response pairs have to be updated regularly and the servers have to use the same algorithm to determine if a nonce is expired. Or a server has to store, for each nonce, when it used this nonce the first time, to challenge the user to be able to detect when this nonce should not be used anymore. In both cases, the backup server can authenticate a user only for a limited amount of time:

$$\text{number of precalculated responses left} * \text{time a nonce is valid} = \text{time a user can be authenticated}$$

In the first case, a lot of bandwidth is needed to keep all backups up to date with valid nonce's and responses. In the second approach, a lot of memory is needed to store

when a nonce was used for the first time. But as this memory can also be disk space or a database, this solution seems to be the better alternative.

In the optimal case, the backup only needs to store the nonce and the response, and a date and time for the pair which is currently in use. This would only be a few bytes for each user of the federation. So the total space required depends very much on the number of the users within the federation, how long a server uses the same nonce for the authentication of a user and how much nonce-response pairs a server stores for each user. Nonce's which have been used and are not valid anymore can simply be deleted from memory, because from the security perspective the reuse of a nonce after its "expire" time would be negligent. This saves memory, and a server cannot authenticate a user anymore if it does not have any nonce-response pairs left for this user.

If an individual expires time for each nonce is stored, it makes much more sense if the backup server notifies the primary server that it needs new nonce-response pairs for a user, instead of the primary server sending nonce-response pairs blindly to the backup servers. This is because in this solution, unused nonce's do not expire and as long as the backup stores them, they can be used. As a side effect, this solution will save a lot of bandwidth because only the actual required number of nonce-response pairs is transmitted between the servers.

The solution is that the backup server sends a NOTIFY directly addressed to the primary, which contains a the new header `User-Responses`. This header contains the username of the user for which precalculated responses are required and the number of desired responses. Note that sending a NOTIFY without receiving a SUBSCRIBE first do not meet the normal event model for SIP [34], but the configuration as primary and backup servers implies the interest of the other server on the "status" change.

An example `User-Responses` header from a NOTIFY to the primary server could look like this:

```
User-Responses: user="alice", responses=5
```

The primary server then transmits the requested nonce-response pairs in the new `User-Nonce-Responses` header included in the 200 OK response back to the backup server. This header contains the username too, to allow that responses for several user can be requested and answered. Furthermore, the header contains the nonce and the response. Finally, it should contain the hash algorithm which was used to create the response, to allow other hash algorithms then the default MD5. If the default MD5 was used the hash algorithm can be omitted to save bytes and parsing effort.

An example of the new `User-Nonce-Responses` header from the 200 OK could look like this:

```
User-Nonce-Responses: username="alice", nonce="592...a32",  
response="3e9...21a", hash-algorithm=MD5
```

Both headers, `User-Responses` and `User-Nonce-Responses`, can appear several times in one message and do not overwrite each other, like the `Authorization` and `Proxy-Authorization` header. This should save resources if responses for several users at the same are requested.

Implementation effort	Low	Trustworthiness	Not required
Management effort	Medium	Former member	No problem
Client effort	Low	Traffic between servers	Can be high
Compatible	Yes	Primary contact	Not required

Table 4.6: Precalculated responses

### 4.3.6 Public Keys as Password

From the current position of cryptography the most secure alternative would be to change the authentication scheme from a shared secret, the password of the user, to a private/public key approach. With this approach the backup servers do not have to be trustful, and former members of the federation are no problem: Knowledge of a user's public key cannot be used to impersonate her.

To avoid the management effort for a complete Public Key Infrastructure (PKI) with signed keys, certificates, key revocation list etc, the easiest solution is to use the public key as a replacement for the password of the user. First the user submits his public key over a web frontend to his primary server (this could be replaced in the future by a standardized SIP method which allows the UA to upload the public key to the server). Then the primary server replicates the public key of the user to all the backup servers. The backup servers, only trust the public keys which they received from the other peers in the federation.

If a user loses his private key or his key is compromised, he just transmits his new public key to his primary server and this overwrites the old key. The primary server then replicates this new public key again to all the other peers. However, this also introduces the chance that a backup could miss the new public key for a user if the outage of this server overlaps with an outage of the primary server.

How the trust relationship is established between the primary server operator and the user in this Public Key (PK) lightweight solution is beyond the scope of this thesis. Also, how a user creates a private and public key pair or how he inserts his private key into his UA will not be examined any further.

One advantage of a PK solution is that a UA does not have to wait for a challenge within a 401 Unauthorized response to authenticate, but can simply sign every outgoing request with its private key. This reduces the traffic between the UA and the server and minimizes the delay to establish a call. But as asymmetric cryptographic algorithms need a lot more processing power than symmetric algorithms, this is not the best solution for small, mobile or battery-powered devices.

Unfortunately this solution is not compatible with existing UA's, but on the other hand, the UA does not need to contact its primary server before it can use any of the backup servers, and the usage of the backup server is not limited in time like in the other solutions.

### 4.3.7 PK's with a Public Key Infrastructure

To eliminate some of the disadvantages of the PK's as password replacement solutions, the Public Keys of the users can be used with a Public Key Infrastructure. This means

Implementation effort	High	Trustworthiness	Not required
Management effort	High	Former member	No problem
Client effort	Medium	Traffic between servers	Medium
Compatible	No	Primary contact	Not required

Table 4.7: PK's without PKI

that each server operator too creates a public private key pair, and uses its private key to sign the public keys of its users. Then the signed keys of all users from a federation are stored at a public key server, and the backup peers are just configured to accept all requests which are signed with a key signed by one of the peers from the federation.

The PKI also provides a Certificate Revocation List (CRL), which lists all keys which are compromised and should not be used anymore. Thus the PKI eliminates the chance that a backup misses a PK update because of an overlapping outage. It also removes the requirement to replicate any information for authentication purposes.

With such a PKI, the backups do not need to store the public keys of all federation users because they can always download the public key of the user from the key server. But this download time adds a big delay to each authentication procedure, which is not desirable in a real time SIP network.

Of course, a server could store the keys of the users to prevent the delay from downloading the key each time. But on the other hand, a server should check each time before using a public key if it is not contained in the current CRL. A check of all stored keys against the CRL can be done periodically but obviously the most secure alternative is to make this check every time a key is used. But as this check introduces a big delay into request processing, the periodic CRL check is the more practical approach.

The advantages of this solution are that the UA's do not need to contact their primary server first, and that the trustfulness of the other peers and former federation members is not a problem. It requires no additional synchronization traffic between the servers to enable the authentication. As a side effect, the keys of the user could also be used to sign traffic bills at the backup peers to enable clearing between the server operators. Authentication at servers outside the federation is also no problem if they can access the public key server.

But the drawbacks are also not insignificant. Firstly, this solution is incompatible with existing UA's. Secondly, the asymmetric cryptography algorithms need more processing power. Thirdly, the management effort to keep the PKI and all the servers in an up-to-date state is very high. And if the whole PKI is not implemented as a geographically distributed highly available solution, then all the servers of the federation will depend on the same instance of the required authentication service, see Section 3.1, and that would be a SPoF.

#### 4.3.8 PK's as Fallback

Finally, one of the Public Key solutions can be combined with the Daily Pass or the pre-calculated responses solution, which eliminates the drawback of the non-PK solutions,

Implementation effort	Very high	Trustworthiness	Not required
Management effort	Very high	Former member	No problem
Client effort	Medium	Traffic between servers	Low
Compatible	No	Primary contact	Not required

Table 4.8: PK's with PKI

Implementation effort	Medium	Trustworthiness	Not required
Management effort	Low	Former member	No problem
Client effort	High	Traffic between servers	Low
Compatible	No	Primary contact	Not required

Table 4.9: PK fallback

which is that they are time limited. This means a UA first tries to use the non-PK approach to authenticate at a backup server, and if the Daily Pass is not valid anymore or if the backup has no precalculated responses left for this user, the UA uses its PK as fallback and authenticates by signing the request or a challenge with its private key. Also, the disadvantage of the increase in the required processing power for the PK solutions only has an effect if the UA needs to switch back to this fallback system.

If PK is combined with Daily Pass, the backup server can even create new Daily Passes for the user, if the latter has successfully authenticated with his PK. For the combination with precalculated responses the UA has to use its PK for authentication until its primary server comes back online.

But both solutions also inherit some of the drawbacks. A management of the PK's is required and that only to serve as a fallback system. Also, both combinations require synchronization traffic between the servers, which is not required if the PK with PKI solutions is chosen. Finally, a combination of two approaches increases the implementation effort, and makes the handling of the authentication procedure a lot more complex.

### 4.3.9 Conclusion

As already stated there is no optimal solution, it mainly depends on

- can you trust the backup servers and former federation members?
- does the solution have to be compatible with the existing UAC's?
- is additional traffic between the servers a problem?

and maybe also on factors which are not considered here, e.g. the grade of security of the used cryptography.

For the implementation part of this thesis we chose the precalculated responses, because storage space is cheap today, the solution is compatible with the existing UAC's, the synchronization traffic can be kept low and the fact that the solution only bypasses a time limited outage of the primary is not too big a penalty.

Solution	Pro	Contra
Contact locations	<ul style="list-style-type: none"> <li>• Very easy to implement</li> <li>• Compatible with existing clients</li> </ul>	<ul style="list-style-type: none"> <li>• Authentication time is limited</li> <li>• Completely insecure</li> </ul>
Replicate H(A1)	<ul style="list-style-type: none"> <li>• Easy to implement</li> <li>• Compatible with existing clients</li> </ul>	<ul style="list-style-type: none"> <li>• Backups have to be trustworthy</li> <li>• Formerly trustworthy backups are a big problem</li> </ul>
Daily Pass for all servers	<ul style="list-style-type: none"> <li>• No password sharing required</li> </ul>	<ul style="list-style-type: none"> <li>• Requires trustworthy backups and formerly trustworthy backups are a problem</li> <li>• Authentication time is limited</li> <li>• Requires first contact with primary</li> <li>• Incompatible with existing clients</li> </ul>
Daily Pass for each server	<ul style="list-style-type: none"> <li>• No password sharing required</li> <li>• Untrustworthy backups can be excluded</li> </ul>	<ul style="list-style-type: none"> <li>• Authentication time is limited</li> <li>• Requires first contact with primary</li> <li>• Incompatible with existing clients</li> </ul>
Precalculated responses	<ul style="list-style-type: none"> <li>• Compatible with existing clients</li> </ul>	<ul style="list-style-type: none"> <li>• Either same nonce algorithm and high sync traffic</li> <li>• Or large memory space required</li> <li>• Authentication time is limited</li> </ul>
Public Keys as passwords	<ul style="list-style-type: none"> <li>• Easier implementation than a complete PKI</li> </ul>	<ul style="list-style-type: none"> <li>• Synchronization between servers</li> <li>• Incompatible with existing clients</li> </ul>
Public Key with PKI	<ul style="list-style-type: none"> <li>• Requires no synchronization</li> <li>• Completely secure</li> </ul>	<ul style="list-style-type: none"> <li>• Slower than local authorization</li> <li>• Big overhead for key management</li> <li>• Incompatible with existing clients</li> </ul>
PK's as fallback	<ul style="list-style-type: none"> <li>• Secure but not always high client effort</li> </ul>	<ul style="list-style-type: none"> <li>• Server synchronization</li> <li>• Very complex</li> <li>• Incompatible with existing clients</li> </ul>

Table 4.10: Authentication schemes overview

## Chapter 5

# Implementation of Replication and Authentication

In this chapter we will give a detailed overview of how we implemented the solutions from the previous chapter for the SIP Express Router (SER). The first Section 5.1 explains how we implemented the replication of the user location database over SIP, then Section 5.2 presents the details of the precalculated responses implementation.

As an introduction, we will now give a brief overview of the SIP Express Router:

The SER is a SIP server implemented in C which can act as proxy, registrar, forwarder and has many extended features like SMS and Jabber gateway, ENUM and Radius support, and an interface for an application server. The whole SER implementation is optimized for maximum speed and great flexibility. The SER is separated into a core which handles message receiving and sending, memory management, request parsing, logging and other basic things. The second part are the modules which are loaded at the startup of the server and add various features to the core. For example, there exists modules for transaction management (TM), for database connectivity (to MySQL, PostgreSQL or plain text files), for a user location database (ULD) and the associated module to use this ULD as registrar. All the extended features from above are implemented as modules which can be loaded optionally.

Each incoming request is handled by a script which configures the behavior of the server. This configuration script contains the "routing logic" for all requests, and is composed of commands from the SER core and commands which the modules exported after their loading. Each of these commands can have one or two parameters, and the command is called with the request and these parameters to determine what happens with the request. For example, the registrar module provides the commands `save` and `lookup`. The `save` command is called to process the content of REGISTER requests and change the content of the ULD accordingly if necessary. The `lookup` command is called for requests like INVITE to look up if a contact for the requested user is available, and to change the destination of the request appropriately.

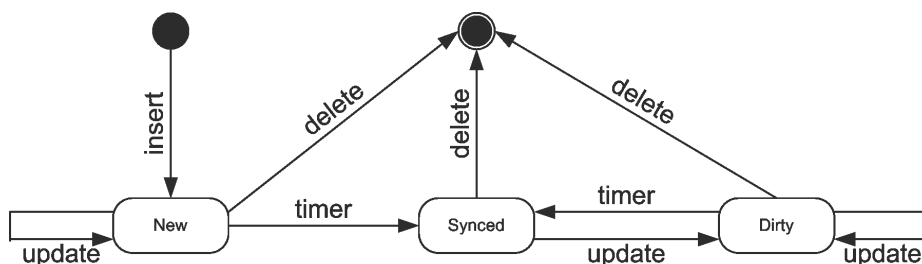


Figure 5.2: The state of a contact without the zombie state

## 5.1 SIP-replication of User Location Database

The existing user location database in the SIP Express Router is organized as follows: each script command to store the contacts of a REGISTER has a domain parameter. This enables the administrator to have different tables for each domain which the proxy serves. For each user which registers in a domain, an Address of Record (AOR) is inserted into the domain, and each of these AOR's can have multiple contacts for the same user. Figure 5.1 gives an overview of the design.

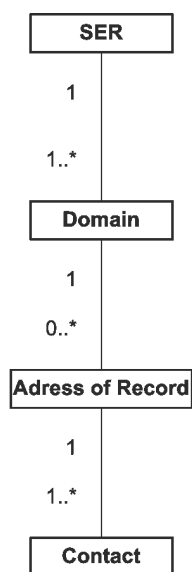


Figure 5.1: Design of the ULD in Memory

The whole database is held in memory to get the best performance. To achieve persistency over restarts of the server, the ULD in memory is synchronized with a real database like MySQL. The administrator can choose between direct synchronization on every change (write through), or a delayed synchronization, where a timer process periodically writes the changes into the database (write back). Although the second alternative gives better performance, the write through scheme should be chosen in an HA environment.

To support the write back scheme in combination with replication we introduced a new zombie state to the existing synchronized and unsynchronized states. A zombie contact is a contact which was removed by the user with a REGISTER request where the Expires header is set to zero. Instead of removing the contact, it is now turned into a zombie to enable the replicator to synchronize this unregistered with the other peers. This implementation results in the fact that we are only able to replicate the last status of each contact. But this does not do any harm, because the last status will overwrite any previous information of the same contact and the information of interim contacts is not of interest. The state of a contact is now also written to the database.

Figure 5.3 illustrates the possible states, and the transitions between the states, after the introduction of the new zombie state for the case when the write back scheme is used for ULD. Before the zombie state introduction, or if the replication module is not loaded, a contact will only be in the states *New*, *Synced* or *Dirty*, as illustrated in Figure 5.2. The state *New* means that this contact is currently only in the memory but not in the

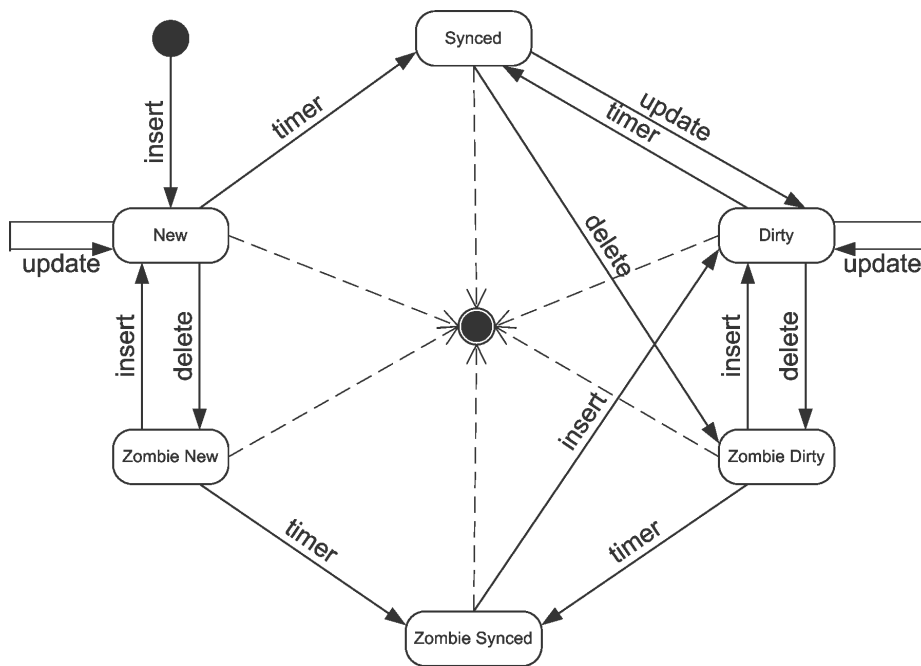


Figure 5.3: The state of a contact after introduction of the zombie state

database yet. *Synced* signifies that this contact is written to the database with exactly these values. And a contact is in the state *Dirty* if the values in the memory differ from the values stored in the database. Because we also store contacts in the zombie state in the database, we also needed three zombie states to be able to reflect the difference between the memory and the database. The arrows to the terminating state are dashed in Figure 5.3, because a contact can be removed from all states if it is not marked for replication anymore (then it will be removed from memory and database immediately).

Turning the normal contact into a zombie instead of storing an extra contact for the unregister for example, also has a drawback. We first have to look at the replication status of the original contact, before we turn the contact into a zombie and mark it for replication. Otherwise we would overwrite the replication flag and maybe replicate the unregister to a peer where we still had not replicated the original REGISTER to. Ultimately, it would not do any harm if we sent an unregister to a peer which had not received the original REGISTER for this contact, but it is completely unnecessary. So we check first the replication flag of the contact for which we received the unregister and identify to which peers we really have to replicate the unregister.

The combination of the new replication mark and the zombie state for every contact enables us to restore the state of the replication on server startup without a need to store any additional information in persistent memory. The replication mark stores, for each contact in a bitmask, the destination to which it still has to be replicated.

Figure 5.4 illustrates that the timer event in the write back scheme now has to decide over three dimensions if a contact should be removed from the database. Before the introduction of the replication mark and zombie state it was just a decision as to whether

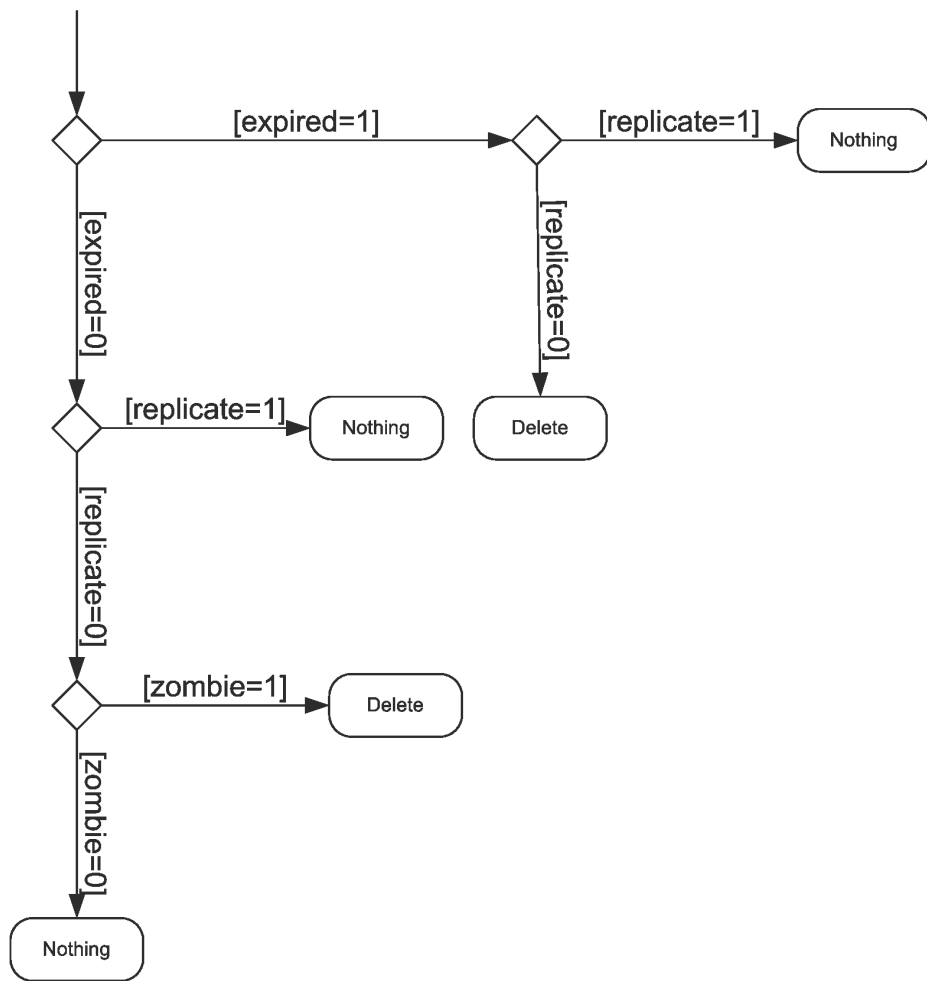


Figure 5.4: Decision tree in the ULD timer event after zombie and replication introduction

the contact was expired or not.

Because we only create a request packet from the given contact values and hand it to the Transaction Management (TM) for delivery, we cannot guarantee that the exact Expires values will be stored at the other peers. We recalculate the expires value for every request we try to replicate, but the TM tries to deliver these requests for a period of time. If it does not succeed directly after we started the transaction but after some seconds, then the receiving peer will store an absolute point of time as an expires value, which is the difference in seconds between the moment we started the transaction and the moment when the peer received it successfully. This is a result of the fact that the expires value is given as a delta to the current time, and absolute time values are deprecated since RFC3261 [36]. But in general this not a problem, because normally the expires value contains at least several minutes and the TM will try to deliver the request only for a few seconds. So the expires value at the other peer can have a maximum

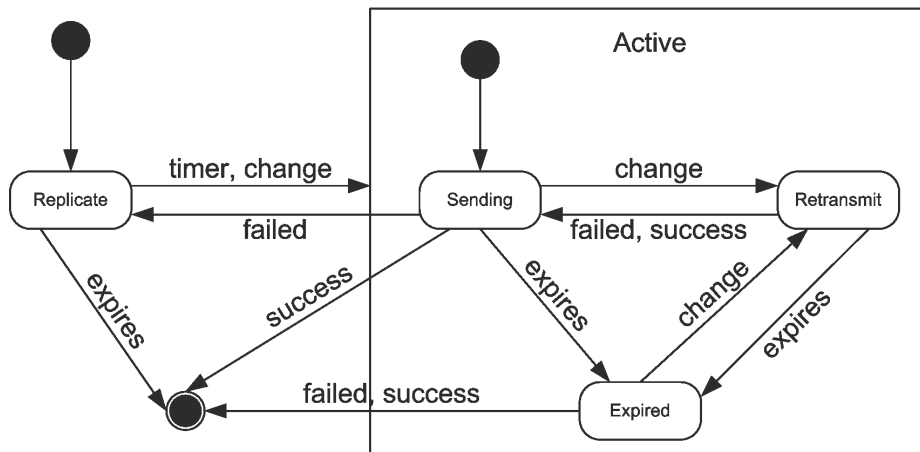


Figure 5.5: States of a contact within replication

difference from the correct value only of the amount of seconds required while the TM is trying to deliver. At worst, the result would be that a peer would look up the contact and forward a request although the UA is not reachable at the contact address anymore, and the request will time out.

The replication function is implemented in a new module for SER. The replicator module is called on every change in the ULD over callbacks. On a callback for a new contact, the replicator inserts the required information into the internal list of the replicator. The replicator needs to store some additional information for every contact to prevent two problems from occurring:

- A contact can expire while TM is trying to deliver. In this case, the contact cannot be removed from the internal replication list because otherwise the callback at the end of the TM function would fail.
- A contact can change while TM is trying to deliver. In this case, we cannot simply start a new transmission try because it would create a race between the already running transmission and the new one. Also, the contact which would be stored at the destination peer would be random.

To prevent these problems, we store an internal state for each contact, diagrammed in Figure 5.5. If the TM function for delivery is called, the contact changes from the normal state to *Active*. A callback for expiration of the contact changes the internal state of the contact to *Expired*, which results in the removal of the contact after the TM function has finished. A callback for the deletion or change of the contact would change the state to *Retransmit* and result in another transmission of the changed contact after the running TM function has finished.

Additionally, the internal list is used to find the first contact which still has to be replicated to one of the configured peers and is currently not in the active state. With this first entry for that peer it will be tested if the peer is responding again. If the transmission of this specially marked first contact is successful, the complete list will be traversed to look for more contacts which have the same destination peer in their replication mark.

## 5.2 Distributed Precalculated Authentication

To use precalculated responses with SER we have to implement two parts. Firstly, we need to be able to distribute precalculated responses and nonce's. Secondly, we need a way to authenticate the users with the precalculated responses instead of the password of the user.

We implemented both parts in a new module for SER which uses the database interface to store all required information in a database. Because we assume that the backup peers will only help out the primary server in the case of a failure, speed is not the most important factor for this friendly turn.

Firstly, we need to inform the backup peers which users are subscribed to the service of the primary server. This means we have to notify the backup peers about new users of the service. If we notify a backup peer about a new user, we additionally provide a set of nonce's with precalculated responses for this user. Thus we can simply use the `User-Nonce-Responses` in a `NOTIFY` to make the backups aware of the new user. To prevent the usage of the backup peers by a user which is no longer subscribed to the service, we also notify the backup peers about removed users. We do not want to introduce a new header just for removing users from the backups databases, so we simply send a `NOTIFY` to the backup with a `User-Nonce-Response` header, which contains the username but an empty nonce and response.

The backup peers regularly check their databases for foreign users for which no precalculated responses are available anymore, or for which the number of precalculated responses fell below a configured threshold. In this case, the backup peer sends a `NOTIFY` to the primary server which contains the `User-Responses` header with the username and the desired number of nonce-response pairs. The primary server responds with `200 OK`, which contains the `User-Nonce-Responses` header with the nonce's and precalculated responses pairs for the user.

The second part of the module contains the new authentication scheme for the backup peers. If a request from a non-local federation user has to be authenticated, the module searches the database for available precalculated responses for this user, and a `401 Unauthorized` reply is generated with the nonce from the database if one is still available for this user. Additionally, this entry in the database is updated with the time and date of the first use of this nonce-response pair.

When a request with an authorization header is received by a backup peer, it looks up the used nonce and user combination in the database. If the entry is present and the date and time value of this entry is not below the configured usage time of nonce's, the responses from the request and the database are compared. Equality of the responses means the request is successfully authenticated. In case the nonce used by the UA in the authorization header is not present at the database anymore or is expired<sup>1</sup>, the server challenges the UA again with one of the remaining nonce's. If no nonce is available for the user any more, the server responds with an error.

Theoretically, the UA could try to use another backup server when the one used server has no nonce's left, but this special "intelligence" is not available in UA's today.

---

<sup>1</sup>The stored first usage time of the nonce plus the configured usage time is below the current time.

The timer event which checks, for users with undercharged nonce-response pairs, also checks for nonce's which are expired. These expired nonce's are deleted from the database.

## Chapter 6

# Validation

Unfortunately, we did not have the chance until the end of this thesis to test our implementation in a real deployment. The Figures from 6.1 to 6.6 illustrate a test run in our labs. The figures are snapshots from the output of the statistic function of the replication modules used over the application server connection. In this test, one server replicated 2000 incoming REGISTER requests to the backup servers *alderan.ohlmeier.de* and *tatoine.ohlmeier.de*.

A brief explanation of the column headers:

**Peer:** the number of the peer from the configuration file.

**State:** the reply code of the last replicated transaction.

**Success:** the number of successfully replicated contacts.

**Pending:** the number of contacts which are currently transmitted by the TM.

**Failed:** the number of transactions which ended with a temporary reply code.

**Errors:** the number of transactions which are replied to with a final error.

**Protocol:** as internal number which is used for replication to this peer.

**Name:** the address of the peer as DNS name or IP address.

Figure 6.1 shows the output of a server that has just started, which has peers configured for replication. In Figure 6.2 the test started and ran normally, 663 contacts were replicated successfully to both peers and no error occurred. Figure 6.3 shows the beginning of an outage of the second peer. Replication of all 2000 contacts to the first server was already completed successfully whereas replication to the second peer was interrupted after the 859th contact. The TM still tried to deliver the remaining 1141 records (*Pending*). The State is still 200 because none of these 1141 deliveries are finished yet. In Figure 6.4 the second peer still had not recovered from its failure, but none of the unsuccessful replications are handled by the TM anymore. The replication module stored as last status of this peer 408, which means the last request to this peer timed

Peer	State	Success	Pending	Failed	Errors	Protocol	Name
1	0	0	0	0	0	1	'alderan.ohlmeier.de'
2	0	0	0	0	0	1	'tatoine.ohlmeier.de'

Figure 6.1: Replication statistics before test start

Peer	State	Success	Pending	Failed	Errors	Protocol	Name
1	200	663	0	0	0	1	'alderan.ohlmeier.de'
2	200	663	0	0	0	1	'tatoine.ohlmeier.de'

Figure 6.2: Replication test runs normally

out without any response. Only one request is pending because with this single request the replication module tests regularly if the peer has recovered. The sum of *Success*, *Pending* and *Failed* is not 2000 because every failed "recovery test" will increase the *Failed* number. After some time the second peer recovers from its failure, which is detected by the replication module in Figure 6.5. The *State* changes from 408 to 200 again and nearly all failed requests are already sent by TM (882 *Pending*). Several seconds after its recovery the second peer is synchronized again with both of the other peers (Figure 6.6).

In our tests we measured a speed of approximately 70 requests per second for the re-synchronization after the recovery of a failed server. So large user populations should also be resynchronized after a few minutes again. However, the improvement of the re-synchronization speed by code optimization and further research is a working item for the future.

The tests with several hardware and software user agents to use this replicated information at the backup servers were not very successful. Only the Cisco 7960 with the newest firmware was able to use the backup servers from the SRV records with all features. Other UA's, for example, switched over on a failure of their primary server to one of the backups, but they never switched back to the primary again. Likewise, ICMP error messages were ignored by the majority of the UA's, although the reaction to these error messages can improve the switch-over time dramatically. The worst SRV implementation of some UA's only looked up their primary server, but completely ignored the information about backup servers.

Peer	State	Success	Pending	Failed	Errors	Protocol	Name
1	200	2000	0	0	0	1	'alderan.ohlmeier.de'
2	200	859	1141	0	0	1	'tatoine.ohlmeier.de'

Figure 6.3: Second peer failed during test

Peer	State	Success	Pending	Failed	Errors	Protocol	Name
1	200	2000	0	0	0	1	'alderan.ohlmeier.de'
2	408	859	1	1142	0	1	'tatoine.ohlmeier.de'

Figure 6.4: Second peer still not recovered

Peer	State	Success	Pending	Failed	Errors	Protocol	Name
1	200	2000	0	0	0	1	'alderan.ohlmeier.de'
2	200	860	882	1142	0	1	'tatoine.ohlmeier.de'

Figure 6.5: Recovery of the second peer detected

Peer	State	Success	Pending	Failed	Errors	Protocol	Name
1	200	2000	0	0	0	1	'alderan.ohlmeier.de'
2	200	2000	0	1142	0	1	'tatoine.ohlmeier.de'

Figure 6.6: All peers are synchronized again

## Chapter 7

# Conclusion

The availability of signaling with SIP on the Internet only reaches 98 percent, which is clearly below the five nines from the competing PSTN. Nowadays HA technology can improve the availability locally, but does not prevent errors in the network between the client and the server of the service provider. The UAC can decide best which hosts it can reach or not, thus we reasoned to place the intelligence into the previous hop. The DNS SRV records provide the mechanism to place the intelligence at the UAC and with full support by the UAC's makes a set up a geographically distributed federation of servers possible.

To permit a federation of loosely coupled servers we designed a SIP-replication, which provides the highest possible flexibility and only duplicates the minimum required information to keep the servers of a federation synchronized. The implementation of replication as a module for the SIP Express Router proves the utilisability of our design, although there is certainly space left for improvements in the implementation.

Unfortunately, there is no ultimate solution to enable the authentication of users at backup servers without giving the passwords to the latter. We provided several possible solutions, all with their specific advantages and drawbacks, allowing a network designer to choose the one that best meets his requirements. The implementation of the precalculated response solution will have to prove its usability in a real deployment.

The Figures 7.1 and 7.2 oppose two examples of a state-of-the-art HA solution and a federation which uses SIP-replication to improve the availability.

To summarize, we believe that a distributed approach, like our federations have, has a high potential to increase the availability of SIP services on the Internet significantly. The implementation of our solution makes the deployment of a federation possible, which proves the superiority of the distributed approach.

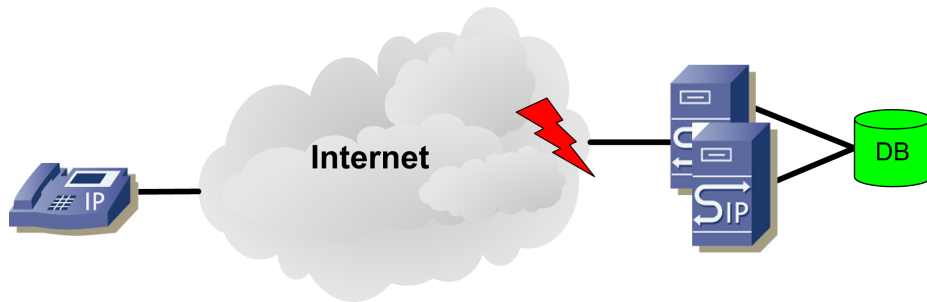


Figure 7.1: An example of a state-of-the-art solution

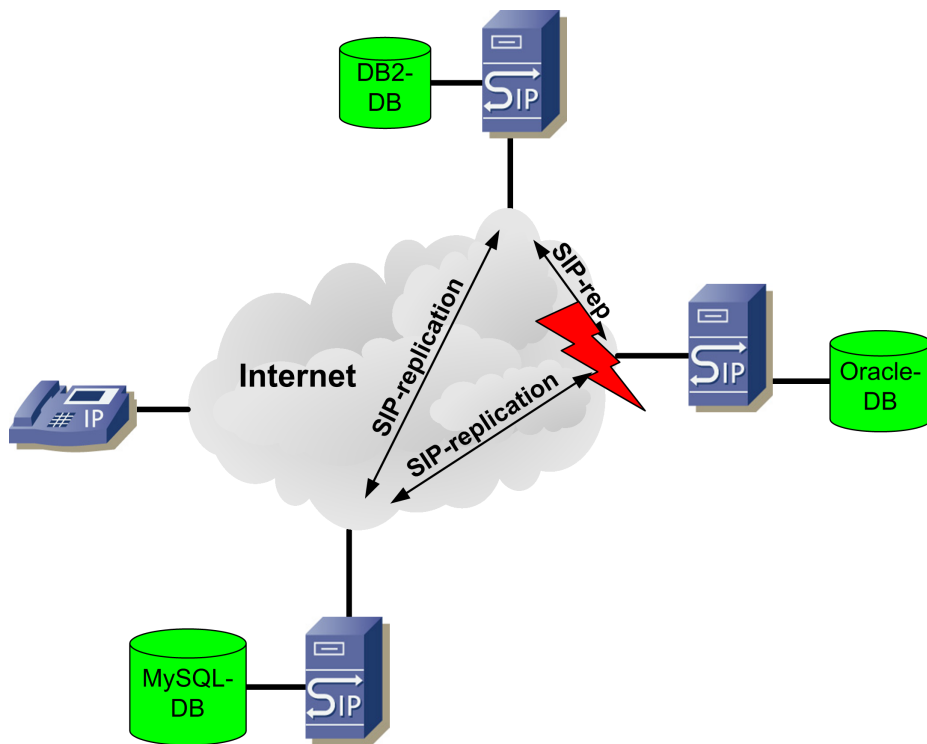


Figure 7.2: A federation with SIP-replication improves availability

# Chapter 8

## Outlook

As already noted the achieved performance of the SIP-replication does not completely meet the high expectations. Thus we will try to improve the throughput of the synchronization after a recovery, for example, by merging the contact updates for one address of record into one update message.

For the future the implementation of one of authentication solutions which are not compatible with existing UA's is also considered.

And the whole implementation will have to prove its useability and performance in real federation deployments in the future.

Two things are very closely related to the work in this thesis: load distribution and peer-to-peer networks. Here we will briefly give some questions, references and ideas for the future work on these items.

### 8.1 Load Distribution

On the Internet, especially for HTTP servers, clusters are often used to combine load distribution and high availability. In principle, this is also possible for SIP servers and is no problem if the SIP servers at the hosts of the cluster are running statelessly. But probably the majority of all SIP servers run statefully, and this causes some problems (in this case the stateless of the HTTP protocol is an advantage).

If a SIP cluster is running statefully all parts of a transaction should be delivered to the same host within the cluster to prevent synchronization of all transaction states between the hosts. One idea could be that the load distributor in front of the cluster calculates a hash value of the most important values of a SIP request (or response) to decide to which host this packet will be forwarded. This also has the advantage that the load distributors in front of the cluster, and there should be at least two to cover the failure of one of them, do not have to be synchronized with each other because each of them should get the same hash result for the same request if they take over the job of a failed one. Still a problem in this case is the failure or removal of a host from the cluster for the transactions which were served by this host.

Additionally, the load distributors could place the content of already parsed header fields into an additional header or header field to prevent the double parsing by the load distributor and the worker backend.

The idea from [17] to distribute the work to different hosts according to the kind of work could be transferred to SIP. So a distribution according to the request method (`INVITE`, `REGISTER`, `MESSAGE`, etc.) could be considered to develop hosts which are specialized and optimized for their job.

A problem with the implementation of the SIP Express Router in a cluster is the memory cache of the user location database. Either the hosts in the cluster have to be synchronized with each other about the changes in their caches, for example, with a version of the SIP-based replication from Section 4.2 optimized for short ranges, or they have to use a shared database backend but without caching the data. With a shared database backend it should be evaluated if the performance improvement of a cluster can compensate for the penalty of making lookups in a remote database instead of the memory.

Finally, TCP connections from UAC's to the cluster should be terminated at the load balancer to prevent problems with dead TCP connections to removed or crashed hosts of the cluster. If TCP or UDP as transport protocol between the load balancer and the cluster hosts is more advantageous have to be evaluated.

To summarize, a SIP load distributor seems to be feasible, but a lot of research and work has to be done to achieve this goal.

## 8.2 Peer-to-Peer

If the size of a federation grows big enough you are not far away from a Peer-to-Peer (P2P) network.

The problems start with how to locate a P2P network or other peers to receive the starting information and to "login" to the network. For a local network the mechanism described in [22] may be suitable, but if the client cannot find another peer in its local network, it will be a little problematic to automatically detect and connect to a P2P network.

After a client successfully connects to a P2P network, the next question is how the user location database is distributed and how lookups for users can be performed. The work on Chord [14, 42, 40] may be helpful, because they developed a mechanism to distribute and redistribute the content of a lookup database reliably over a P2P network. On the other hand, one big P2P network for all the SIP users in the world is hardly imaginable, so the work from Transis [2, 31] may be inspiring because they investigated big local groups which use multicast or broadcast in the local network for communication, and these "communications domains" or "broadcast domains" [2] communicate over logical end-to-end connections with each other. Possibly this could lead to P2P networks in companies which are logically interconnected over end-to-end links.

An interesting point is how authentication can be handled in a P2P network if a central authentication service is missing. Without authentication, accounting too seems to be impossible. Maybe PK's with a PKI can also solve this issue over the borders of a local P2P network.

In this area a lot of research has to be done too before a reasonable solution can be reached.

# Bibliography

- [1] L. Esibov A. Gulbrandsen, P. Vixie. A dns rr for specifying the location of services (dns srv). Request for Comments 2782, February 2000. <ftp://ftp.ietf.org/rfc/rfc2782.txt>.
- [2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication subsystem for high availability. In *FTCS-22: 22nd International Symposium on Fault Tolerant Computing*, pages 76–84, Boston, Massachusetts, 1992. IEEE Computer Society Press.
- [3] AT&T. About AT&T, 2002. <http://www.att.com/network/standrd.html#np>.
- [4] Gary Audin. Reality check on five-nines. *Buisness Communications Review*, May 2002. <http://www.bcr.com/bcsmag/2002/05/p22.asp>.
- [5] H.E. Bal, M.F. Kaashoek, A.S. Tanenbaum, and J. Jansen. Replication Techniques for Speeding up Parallel Applications on Distributed Systems. *Concurrency Practice & Experience*, 4(5):337–355, August 1992.
- [6] W. Milliken C. Partridge, T. Mendez. Host anycasting service. Request for Comments 1546, Novmeber 1993. <ftp://ftp.ietf.org/rfc/rfc1546.txt>.
- [7] Y. Chawathe and E. Brewer. System support for scalable and fault tolerant internet service. In *IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, September 1998.
- [8] Cisco. High-availability solutions for sip enabled voice-over-ip networks, 2002. White Paper.
- [9] CNN.com. Technology - cyber-attacks batter web heavyweights, February 2000. <http://www.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/>.
- [10] Federal Communications Commission. Electronic ARMIS filing system - ARMIS main menu. <http://svartifoss2.fcc.gov/eafs/MainMenu.cfm>.
- [11] Ubiquity Software Corporation. Ubiquity software, 2003. [http://www.ubiquity.net/products/SIP\\_Application\\_Server.php](http://www.ubiquity.net/products/SIP_Application_Server.php).
- [12] Flaviu Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, 1991.

- [13] Flaviu Cristian. Abstractions for fault-tolerance. In Karen Duncan and Karl Krueger, editors, *Proceedings of the IFIP 13th World Computer Congress. Volume 3 : Linkage and Developing Countries*, pages 278–286, Amsterdam, The Netherlands, 1994. Elsevier Science Publishers.
- [14] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with chord, a distributed lookup service. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schloss Elmau, Germany, May 2001. IEEE Computer Society.
- [15] Sean Donelan. Internet outage trends. Atlanta, Georgia, 2001. NANOG 21.
- [16] Hal Stern Evan Marcus. *Blueprints for High Availability: Designing Resilient Distributed Systems*. Wiley, 2000.
- [17] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-based scalable network services. In *Symposium on Operating Systems Principles*, pages 78–91, 1997.
- [18] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. Http authentication: Basic and digest access authentication. Request for Comments 2617, June 1999. <ftp://ftp.ietf.org/rfc/rfc2617.txt>.
- [19] IETF. Requirement for internet hosts – application support. Request for Comments 1123, October 1989. <ftp://ftp.ietf.org/rfc/rfc1123.txt>.
- [20] ITU-T. *H.323 Annex R: Robustness Methods for H.323 Entities*, 2002.
- [21] ITU-T. *H.323 draft V5 - Packet-based multimedia communications systems*, 2002.
- [22] J. Rosenberg J. Kempf. Finding a sip server with slp. <http://www.jdrosen.net/ietf.html>, February 2000.
- [23] H. Schulzrinne J. Rosenberg. Session initiation protocol (sip): Locating sip servers. Request for Comments 3263, June 2002. <ftp://ftp.ietf.org/rfc/rfc3263.txt>.
- [24] J. Janak. Sip proxy server effectiveness. Master’s thesis, Czech Technical University, June 2003.
- [25] Wenyu Jiang and Henning Schulzrinne. Assessment of voip service availability in the current internet. In *Passive & Active Measurement Workshop*, San Diego, CA, April 2003.
- [26] Sanjay Kalra. Carrier class availability for ip networks, October 2002. Juniper Networks.
- [27] Jussi Kangasharju and Keith W. Ross. A replicated architecture for the domain name system. In *INFOCOM (2)*, pages 660–669, 2000.
- [28] I. Keidar. A highly available paradigm for consistent object replication. Master’s thesis, Institute of Computer Science, The Hebrew University of Jerusalem, 1994.
- [29] S. Knight, D. Weaver, D. Whipple, R. Hinden, D. Mitzel, P. Hunt, P. Higginson, M. Shand, and A. Lindem. Virtual router redundancy protocol. Request for Comments 2338, April 1998. <ftp://ftp.ietf.org/rfc/rfc2338.txt>.

- [30] H. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-hashing for message authentication. Request for Comments 2104, February 1997. <ftp://ftp.ietf.org/rfc/rfc2104.txt>.
- [31] D. Malki, Y. Amir, D. Dolev, and S. Kramer. The Transis Approach to High Availability Cluster Communication. Technical Report CS94-14, 1994.
- [32] PacketCable. Voip availability and reliability model for the packetcable architecture, 2000.
- [33] Robbert Van Renesse, Kenneth P. Birman, Bradford B. Glade, Katie Guo, Mark Hayden, Takako Hickey, Dalia Malki, Alex Vaysburd, and Werner Vogels. Horus: A flexible group communications system. Technical Report TR95-1500, 23, 1995.
- [34] A. B. Roach. Session initiation protocol (sip)-specific event notification. Request for Comments 3265, June 2002. <ftp://ftp.ietf.org/rfc/rfc3265.txt>.
- [35] J. Rosenberg. Reconstituting call state in sip user agents. June 2001. [http://www.jdrosen.net/sip\\_recons.html](http://www.jdrosen.net/sip_recons.html).
- [36] J. Rosenberg, H. Schulzrine, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol. Request for Comments 3261, June 2002. <ftp://ftp.ietf.org/rfc/rfc3261.txt>.
- [37] M. Schleifer S. Woolf. Anycasting f.root-servers.net. Phoenix, Arizona, 2003. NANOG 27.
- [38] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley, second edition, 1996.
- [39] Dawn X. Song and Adrian Perrig. Advanced and authenticated marking schemes for IP traceback. In *Proceedings IEEE Infocomm 2001*, 2001.
- [40] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
- [41] Andrew Tanenbaum and Marten van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.
- [42] Tina Tyan. A case study of server selection. Master’s thesis, Massachusetts Institute of Technology, September 2001.

# Appendix A

## Acronyms

**2PC** Two Phase Commit (Protocol)

**3GPP** 3rd Generation Partnership Project

**3PC** Three Phase Commit (Protocol)

**AOR** Address of Record

**ASCII** American Standard Code for Information Interchange

**CRL** Certificate Revocation List

**DB** Database

**DDoS** Distributed Denial of Service

**DigestAuth** HTTP Digest Access Authentication

**DNS** Domain Name System

**GSM** Global System for Mobile communication

**HA** High Availability

**HTTP** Hyper Text Transfer Protocol

**ICMP** Internet Control Message Protocol

**IPSec** Internet Protocol Security

**IPv4** Internet Protocol Version 4

**ISDN** Integrated Services Digital Network

**ISP** Internet Service Provider

**MAC** Media Access Control

**MAC** Message Authentication Code

**P2P** Peer-to-Peer  
**PK** Public Key  
**PKI** Public Key Infrastructure  
**PSTN** Public Switched Telephone Network  
**qop** Quality of Protection  
**rserpool** Reliable Server Pooling  
**SER** SIP Express Router  
**SIP** Session Initiation Protocol  
**SPoF** Single Point of Failure  
**TCO** Total Cost of Ownership  
**TLS** Transport Level Security  
**TM** Transaction Management  
**UA** User Agent  
**UAC** User Agent Client  
**UAS** User Agent Server  
**ULD** User Location Database  
**WAN** Wide Area Network

# Appendix B

## Call Flow Messages

Here we present sample messages for all call flows from this thesis. The content of header fields in the messages are complete fiction. Message bodies are omitted. Too long lines, like WWW-Authenticate and Authorization, are indented with two spaces.

### B.1 Figure 4.1

F1, F2 and F3:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3789 REGISTER
Call-ID: 8940-3829-194-3492@1.2.3.4
Max-Forwards: 10
Contact: <sip:alice@1.2.3.4:5060>
Expires: 600
Content-Length: 0
```

F4:

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3789 REGISTER
Call-ID: 8940-3829-194-3492@1.2.3.4
Contact: <sip:alice@1.2.3.4:5060>;q=0.0;expires=600
WWW-Authenticate: Digest realm="example.com", domain="sip:example.com",
    qop="auth", nonce="f84f1cec41e6cbe5aea9c8e88d359", algorithm=MD5
```

F5:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3789 REGISTER
Call-ID: 8940-3829-194-3492@1.2.3.4
Max-Forwards: 10
Contact: <sip:alice@1.2.3.4:5060>
Expires: 600
Content-Length: 0
Authorization: Digest username="a", realm="example.com",
    nonce="c60f3082ee1212b402a21831ae",
    response="245f23415f11432b3434341c022"
```

F6:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3789 REGISTER
Call-ID: 8940-3829-194-3492@1.2.3.4
Contact: <sip:alice@1.2.3.4:5060>;q=0.0;expires=600
```

## B.2 Figure 4.2

F1:

```
INVITE sip:b@foobar.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>;tag=38f63c-194
To: "Bob" <sip:b@foobar.com>
CSeq: 3790 INVITE
Call-ID: 3741-3942-195-2682@1.2.3.4
Max-Forwards: 10
Contact: <sip:alice@1.2.3.4:5060>
Content-Type: application/sdp
Content-Length: 233
```

F2:

```
SIP/2.0 100 trying -- your call is important to us
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>;tag=38f63c-194
```

To: "Bob" <sip:b@foobar.com>  
CSeq: 3790 INVITE  
Call-ID: 3741-3942-195-2682@1.2.3.4  
Content-Length: 0

**F3, F4 and F5:**

INVITE sip:b@foobar.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1;branch=376E852  
Via: SIP/2.0/UDP 1.2.3.4  
Record-Route: <sip:1.2.3.1>  
From: "Alice" <sip:a@example.com>;tag=38f63c-194  
To: "Bob" <sip:b@foobar.com>  
CSeq: 3790 INVITE  
Call-ID: 3741-3942-195-2682@1.2.3.4  
Max-Forwards: 9  
Contact: <sip:alice@1.2.3.4:5060>  
Content-Type: application/sdp  
Content-Length: 233

**F6:**

SIP/2.0 100 trying -- your call is important to us  
Via: SIP/2.0/UDP 1.2.3.1;branch=376E852  
Via: SIP/2.0/UDP 1.2.3.4  
From: "Alice" <sip:a@example.com>;tag=38f63c-194  
To: "Bob" <sip:b@foobar.com>  
CSeq: 3790 INVITE  
Call-ID: 3741-3942-195-2682@1.2.3.4  
Content-Length: 0

**F7:**

INVITE sip:bob@4.3.2.15:5060 SIP/2.0  
Via: SIP/2.0/UDP 4.3.2.1;branch=279A312  
Via: SIP/2.0/UDP 1.2.3.1;branch=376E852  
Via: SIP/2.0/UDP 1.2.3.4  
Record-Route: <sip:4.3.2.1>  
Record-Route: <sip:1.2.3.1>  
From: "Alice" <sip:a@example.com>;tag=38f63c-194  
To: "Bob" <sip:b@foobar.com>  
CSeq: 3790 INVITE  
Call-ID: 3741-3942-195-2682@1.2.3.4  
Max-Forwards: 8  
Contact: <sip:alice@1.2.3.4:5060>  
Content-Type: application/sdp  
Content-Length: 233

F8:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 4.3.2.1;branch=279A312
Via: SIP/2.0/UDP 1.2.3.1;branch=376E852
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:4.3.2.1>
Record-Route: <sip:1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=38f63c-194
To: "Bob" <sip:b@foobar.com>;tag=8294d34-10a4
CSeq: 3790 INVITE
Call-ID: 3741-3942-195-2682@1.2.3.4
Contact: <sip:bob@4.3.2.15:5060>
Content-Type: application/sdp
Content-Length: 201
```

F9:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.1;branch=376E852
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:4.3.2.1>
Record-Route: <sip:1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=38f63c-194
To: "Bob" <sip:b@foobar.com>;tag=8294d34-10a4
CSeq: 3790 INVITE
Call-ID: 3741-3942-195-2682@1.2.3.4
Contact: <sip:bob@4.3.2.15:5060>
Content-Type: application/sdp
Content-Length: 201
```

F10:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:4.3.2.1>
Record-Route: <sip:1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=38f63c-194
To: "Bob" <sip:b@foobar.com>;tag=8294d34-10a4
CSeq: 3790 INVITE
Call-ID: 3741-3942-195-2682@1.2.3.4
Contact: <sip:bob@4.3.2.15:5060>
Content-Type: application/sdp
Content-Length: 201
```

F11:

ACK sip:b@foobar.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.4  
Route: <sip:4.3.2.1>  
From: "Alice" <sip:a@example.com>;tag=38f63c-194  
To: "Bob" <sip:b@foobar.com>;tag=8294d34-10a4  
CSeq: 3790 ACK  
Call-ID: 3741-3942-195-2682@1.2.3.4  
Content-Length: 0

F12:

ACK sip:b@foobar.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1;branch=376E852  
Via: SIP/2.0/UDP 1.2.3.4  
From: "Alice" <sip:a@example.com>;tag=38f63c-194  
To: "Bob" <sip:b@foobar.com>;tag=8294d34-10a4  
CSeq: 3790 ACK  
Call-ID: 3741-3942-195-2682@1.2.3.4  
Content-Length: 0

F13:

ACK sip:bob@4.3.2.15:5060 SIP/2.0  
Via: SIP/2.0/UDP 4.3.2.1;branch=279A312  
Via: SIP/2.0/UDP 1.2.3.1;branch=376E852  
Via: SIP/2.0/UDP 1.2.3.4  
From: "Alice" <sip:a@example.com>;tag=38f63c-194  
To: "Bob" <sip:b@foobar.com>;tag=8294d34-10a4  
CSeq: 3790 ACK  
Call-ID: 3741-3942-195-2682@1.2.3.4  
Content-Length: 0

## B.3 Figure 4.3

F1:

INVITE sip:b@example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.4  
From: "Alice" <sip:a@example.com>;tag=573f8a-1c76  
To: "Bob" <sip:b@example.com>  
CSeq: 3791 INVITE  
Call-ID: 3798-386f-182-a482@1.2.3.4  
Max-Forwards: 10  
Contact: <sip:alice@1.2.3.4:5060>  
Content-Type: application/sdp  
Content-Length: 233

F2:

```
INVITE sip:bob@128.2.3.5:5060 SIP/2.0
Via: SIP/2.0/UDP 1.2.3.1
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=573f8a-1c76
To: "Bob" <sip:b@example.com>
CSeq: 3791 INVITE
Call-ID: 3798-386f-182-a482@1.2.3.4
Max-Forwards: 9
Contact: <sip:alice@1.2.3.4:5060>
Content-Type: application/sdp
Content-Length: 233
```

F3:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.1
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=573f8a-1c76
To: "Bob" <sip:b@foobar.com>;tag=18392-f829d
CSeq: 3791 INVITE
Call-ID: 3798-386f-182-a482@1.2.3.4
Contact: <sip:bob@128.2.3.5:5060>
Content-Type: application/sdp
Content-Length: 201
```

F4:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=573f8a-1c76
To: "Bob" <sip:b@foobar.com>;tag=18392-f829d
CSeq: 3791 INVITE
Call-ID: 3798-386f-182-a482@1.2.3.4
Contact: <sip:bob@128.2.3.5:5060>
Content-Type: application/sdp
Content-Length: 201
```

F5:

```
ACK sip:b@example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>;tag=573f8a-1c76
```

To: "Bob" <sip:b@foobar.com>;tag=18392-f829d  
CSeq: 3791 ACK  
Call-ID: 3798-386f-182-a482@1.2.3.4  
Content-Length: 0

F6:

ACK sip:bob@128.2.3.5:5060 SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1  
Via: SIP/2.0/UDP 1.2.3.4  
From: "Alice" <sip:a@example.com>;tag=573f8a-1c76  
To: "Bob" <sip:b@foobar.com>;tag=18392-f829d  
CSeq: 3791 ACK  
Call-ID: 3798-386f-182-a482@1.2.3.4  
Content-Length: 0

## B.4 Figure 4.4

F1:

INVITE sip:b@example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.4  
From: "Alice" <sip:a@example.com>;tag=39c72-29f3  
To: "Bob" <sip:b@example.com>  
CSeq: 3792 INVITE  
Call-ID: 295f-1954-ab34-193@1.2.3.4  
Max-Forwards: 10  
Contact: <sip:alice@1.2.3.4:5060>  
Content-Type: application/sdp  
Content-Length: 233

F2:

INVITE sip:bob@128.2.3.5:5060 SIP/2.0  
Via: SIP/2.0/UDP 1.2.2.1;branch=274F6E  
Via: SIP/2.0/UDP 1.2.3.1;branch=274F6E  
Via: SIP/2.0/UDP 1.2.3.4  
Record-Route: <sip:1.2.2.1;unicast=1.2.3.1>  
From: "Alice" <sip:a@example.com>;tag=39c72-29f3  
To: "Bob" <sip:b@example.com>  
CSeq: 3792 INVITE  
Call-ID: 295f-1954-ab34-193@1.2.3.4  
Max-Forwards: 9  
Contact: <sip:alice@1.2.3.4:5060>  
Content-Type: application/sdp  
Content-Length: 233

F3:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.2.1;branch=274F6E
Via: SIP/2.0/UDP 1.2.3.1;branch=274F6E
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:1.2.2.1;unicast=1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=39c72-29f3
To: "Bob" <sip:b@foobar.com>;tag=93f72-45a2
CSeq: 3792 INVITE
Call-ID: 295f-1954-ab34-193@1.2.3.4
Contact: <sip:bob@128.2.3.5:5060>
Content-Type: application/sdp
Content-Length: 201
```

F4:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.1;branch=274F6E
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:1.2.2.1;unicast=1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=39c72-29f3
To: "Bob" <sip:b@foobar.com>;tag=93f72-45a2
CSeq: 3792 INVITE
Call-ID: 295f-1954-ab34-193@1.2.3.4
Contact: <sip:bob@128.2.3.5:5060>
Content-Type: application/sdp
Content-Length: 201
```

F5:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:1.2.2.1;unicast=1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=39c72-29f3
To: "Bob" <sip:b@foobar.com>;tag=93f72-45a2
CSeq: 3792 INVITE
Call-ID: 295f-1954-ab34-193@1.2.3.4
Contact: <sip:bob@128.2.3.5:5060>
Content-Type: application/sdp
Content-Length: 201
```

F6:

```
ACK sip:b@example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
Route: <sip:1.2.2.1;unicast=1.2.3.1>
```

From: "Alice" <sip:a@example.com>;tag=39c72-29f3  
To: "Bob" <sip:b@foobar.com>;tag=93f72-45a2  
CSeq: 3792 ACK  
Call-ID: 295f-1954-ab34-193@1.2.3.4  
Content-Length: 0

F7:

ACK sip:bob@128.2.3.5:5060 SIP/2.0  
Via: SIP/2.0/UDP 1.2.2.1;branch=274F6E  
Via: SIP/2.0/UDP 1.2.3.1;branch=274F6E  
Via: SIP/2.0/UDP 1.2.3.4  
From: "Alice" <sip:a@example.com>;tag=39c72-29f3  
To: "Bob" <sip:b@foobar.com>;tag=93f72-45a2  
CSeq: 3792 ACK  
Call-ID: 295f-1954-ab34-193@1.2.3.4  
Content-Length: 0

## B.5 Figure 4.5

F1:

INVITE sip:b@example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.4  
From: "Alice" <sip:a@example.com>;tag=9432-56c5b3  
To: "Bob" <sip:b@example.com>  
CSeq: 3793 INVITE  
Call-ID: 8392f-548a-391f-38e4@1.2.3.4  
Max-Forwards: 10  
Contact: <sip:alice@1.2.3.4:5060>  
Content-Type: application/sdp  
Content-Length: 233

F2:

INVITE sip:bob@128.2.3.5:5060 SIP/2.0  
Via: SIP/2.0/UDP 1.2.2.1;branch=384F28E  
Via: SIP/2.0/UDP 1.2.3.1;branch=384F28E  
Via: SIP/2.0/UDP 1.2.3.4  
Record-Route: <sip:1.2.2.1;unicast=1.2.3.1>  
From: "Alice" <sip:a@example.com>;tag=9432-56c5b3  
To: "Bob" <sip:b@example.com>  
CSeq: 3793 INVITE  
Call-ID: 8392f-548a-391f-38e4@1.2.3.4  
Max-Forwards: 9  
Contact: <sip:alice@1.2.3.4:5060>  
Content-Type: application/sdp  
Content-Length: 233

F3:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.2.1;branch=384F28E
Via: SIP/2.0/UDP 1.2.3.1;branch=384F28E
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:1.2.2.1;unicast=1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=9432-56c5b3
To: "Bob" <sip:b@foobar.com>;tag=832f9a-47c9f
CSeq: 3793 INVITE
Call-ID: 8392f-548a-391f-38e4@1.2.3.4
Contact: <sip:bob@128.2.3.5:5060>
Content-Type: application/sdp
Content-Length: 201
```

F4:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.4
Record-Route: <sip:1.2.2.1;unicast=1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=9432-56c5b3
To: "Bob" <sip:b@foobar.com>;tag=832f9a-47c9f
CSeq: 3793 INVITE
Call-ID: 8392f-548a-391f-38e4@1.2.3.4
Contact: <sip:bob@128.2.3.5:5060>
Content-Type: application/sdp
Content-Length: 201
```

F5:

```
ACK sip:b@example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
Route: <sip:1.2.2.1;unicast=1.2.3.1>
From: "Alice" <sip:a@example.com>;tag=9432-56c5b3
To: "Bob" <sip:b@foobar.com>;tag=832f9a-47c9f
CSeq: 3793 ACK
Call-ID: 8392f-548a-391f-38e4@1.2.3.4
Content-Length: 0
```

F6:

```
ACK sip:bob@128.2.3.5:5060 SIP/2.0
Via: SIP/2.0/UDP 1.2.2.1;branch=384F28E
Via: SIP/2.0/UDP 1.2.3.2;branch=384F28E
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>;tag=9432-56c5b3
To: "Bob" <sip:b@foobar.com>;tag=832f9a-47c9f
CSeq: 3793 ACK
Call-ID: 8392f-548a-391f-38e4@1.2.3.4
Content-Length: 0
```

## B.6 Figure 4.6

F1:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3794 REGISTER
Call-ID: 3829-29f6-ac13-56fb@1.2.3.4
Max-Forwards: 10
Contact: <sip:alice@1.2.3.4:5060>
Expires: 600
Content-Length: 0
```

F2:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.1;branch=837F45A
From: <sip:a@example.com>
To: <sip:a@example.com>
CSeq: 3794 REGISTER
Call-ID: 3829-29f6-ac13-56fb@1.2.3.4
Max-Forwards: 10
Contact: <sip:alice@1.2.3.4:5060>;q=0.0;expires=600
Content-Length: 0
```

F3:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3794 REGISTER
Call-ID: 3829-29f6-ac13-56fb@1.2.3.4
Contact: <sip:alice@1.2.3.4:5060>;q=0.0;expires=600
```

F4:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.1;branch=837F45A
From: <sip:a@example.com>
To: <sip:a@example.com>
CSeq: 3794 REGISTER
Call-ID: 3829-29f6-ac13-56fb@1.2.3.4
Contact: <sip:alice@1.2.3.4:5060>;q=0.0;expires=600
```

F5:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.5
From: "Bob" <sip:b@example.com>
To: "Bob" <sip:b@example.com>
CSeq: 3795 REGISTER
Call-ID: 3829-29f6-ac13-56fb@1.2.3.5
Max-Forwards: 10
Contact: <sip:bob@1.2.3.5:5060>
Expires: 600
Content-Length: 0
```

F6:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.1;branch=3891EF
From: <sip:b@example.com>
To: <sip:b@example.com>
CSeq: 3795 REGISTER
Call-ID: 3829-29f6-ac13-56fb@1.2.3.5
Max-Forwards: 10
Contact: <sip:bob@1.2.3.5:5060>;q=0.0;expires=600
Content-Length: 0
```

F7:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.5
From: "Bob" <sip:b@example.com>
To: "Bob" <sip:b@example.com>
CSeq: 3795 REGISTER
Call-ID: 3829-29f6-ac13-56fb@1.2.3.5
Contact: <sip:bob@1.2.3.5:5060>;q=0.0;expires=600
```

F8:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.1;branch=2951E2
From: <sip:b@example.com>
To: <sip:b@example.com>
CSeq: 3795 REGISTER
Call-ID: 3829-29f6-ac13-56fb@1.2.3.5
Max-Forwards: 10
Contact: <sip:bob@1.2.3.5:5060>;q=0.0;expires=585
Content-Length: 0
```

F9:

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.6  
From: "Clair" <sip:c@example.com>  
To: "Clair" <sip:c@example.com>  
CSeq: 3796 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.6  
Max-Forwards: 10  
Contact: <sip:clair@1.2.3.6:5060>  
Expires: 0  
Content-Length: 0

**F10:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1;branch=82E4A1  
From: <sip:c@example.com>  
To: <sip:c@example.com>  
CSeq: 3796 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.6  
Max-Forwards: 10  
Contact: <sip:clair@1.2.3.6:5060>;q=0.0;expires=0  
Content-Length: 0

**F11:**

SIP/2.0 200 OK  
Via: SIP/2.0/UDP 1.2.3.6  
From: "Clair" <sip:c@example.com>  
To: "Clair" <sip:c@example.com>  
CSeq: 3796 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.6

**F12:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1;branch=3951A  
From: <sip:b@example.com>  
To: <sip:b@example.com>  
CSeq: 3795 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.5  
Max-Forwards: 10  
Contact: <sip:bob@1.2.3.5:5060>;q=0.0;expires=570  
Content-Length: 0

**F13:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1;branch=93AB451  
From: <sip:b@example.com>  
To: <sip:b@example.com>  
CSeq: 3795 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.5  
Max-Forwards: 10  
Contact: <sip:bob@1.2.3.5:5060>;q=0.0;expires=555  
Content-Length: 0

**F14:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.7  
From: "Dave" <sip:d@example.com>  
To: "Dave" <sip:d@example.com>  
CSeq: 3797 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.7  
Max-Forwards: 10  
Contact: <sip:dave@1.2.3.7:5060>  
Expires: 600  
Content-Length: 0

**F15:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1;branch=99AC123  
From: <sip:d@example.com>  
To: <sip:d@example.com>  
CSeq: 3797 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.7  
Max-Forwards: 10  
Contact: <sip:dave@1.2.3.7:5060>;q=0.0;expires=600  
Content-Length: 0

**F16:**

SIP/2.0 200 OK  
Via: SIP/2.0/UDP 1.2.3.7  
From: "Dave" <sip:d@example.com>  
To: "Dave" <sip:d@example.com>  
CSeq: 3797 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.7  
Contact: <sip:dave@1.2.3.7:5060>;q=0.0;expires=600

**F17:**

SIP/2.0 200 OK  
Via: SIP/2.0/UDP 1.2.3.1;branch=99AC123  
From: <sip:d@example.com>  
To: <sip:d@example.com>  
CSeq: 3797 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.7  
Contact: <sip:dave@1.2.3.7:5060>;q=0.0;expires=600

**F18:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1;branch=31EC839  
From: <sip:b@example.com>  
To: <sip:b@example.com>  
CSeq: 3795 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.5  
Max-Forwards: 10  
Contact: <sip:bob@1.2.3.5:5060>;q=0.0;expires=547  
Content-Length: 0

**F19:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.1;branch=91258F  
From: <sip:c@example.com>  
To: <sip:c@example.com>  
CSeq: 3796 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.6  
Max-Forwards: 10  
Contact: <sip:clair@1.2.3.6:5060>;q=0.0;expires=0  
Content-Length: 0

**F20:**

SIP/2.0 200 OK  
Via: SIP/2.0/UDP 1.2.3.1;branch=31EC839  
From: <sip:b@example.com>  
To: <sip:b@example.com>  
CSeq: 3795 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.5  
Contact: <sip:bob@1.2.3.5:5060>;q=0.0;expires=547

**F21:**

SIP/2.0 200 OK  
Via: SIP/2.0/UDP 1.2.3.1;branch=91258F  
From: <sip:c@example.com>  
To: <sip:c@example.com>  
CSeq: 3796 REGISTER  
Call-ID: 3829-29f6-ac13-56fb@1.2.3.6

## B.7 Figure 4.7

F1, F2 and F3:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3789 REGISTER
Call-ID: 8940-3829-194-3492@1.2.3.4
Max-Forwards: 10
Contact: <alice@1.2.3.4:5060>
Expires: 600
Content-Length: 0
```

F4:

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3789 REGISTER
Call-ID: 8940-3829-194-3492@1.2.3.4
Contact: <sip:alice@1.2.3.4:5060>;q=0.0;expires=600
WWW-Authenticate: Digest realm="example.com", domain="sip:example.com",
    qop="auth", nonce="f84f1cec41e6cbe5aea9c8e88d359", algorithm=MD5
```

F5:

```
REGISTER sip:example.com SIP/2.0
Via: SIP/2.0/UDP 1.2.3.4
From: "Alice" <sip:a@example.com>
To: "Alice" <sip:a@example.com>
CSeq: 3789 REGISTER
Call-ID: 8940-3829-194-3492@1.2.3.4
Max-Forwards: 10
Contact: <sip:alice@1.2.3.4:5060>
Expires: 600
Content-Length: 0
Authorization: Digest username="a", realm="example.com",
    nonce="c60f3082ee1212b402a21831ae",
    response="245f23415f11432b3434341c022"
```

F6:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 1.2.3.4
```

From: "Alice" <sip:a@example.com>  
To: "Alice" <sip:a@example.com>  
CSeq: 3789 REGISTER  
Call-ID: 8940-3829-194-3492@1.2.3.4  
Contact: <alice@1.2.3.4:5060>;q=0.0;expires=600

**F7:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.2  
From: "Alice" <sip:a@example.com>  
To: "Alice" <sip:a@example.com>  
CSeq: 3789 REGISTER  
Call-ID: 8940-3829-194-3492@1.2.3.4  
Max-Forwards: 10  
Contact: <alice@1.2.3.4:5060>;q=0.0;expire=585  
Content-Length: 0

**F8:**

REGISTER sip:example.com SIP/2.0  
Via: SIP/2.0/UDP 1.2.3.2  
From: "Alice" <sip:a@example.com>  
To: "Alice" <sip:a@example.com>  
CSeq: 3789 REGISTER  
Call-ID: 8940-3829-194-3492@1.2.3.4  
Max-Forwards: 10  
Contact: <alice@1.2.3.4:5060>;q=0.0;expire=570  
Content-Length: 0

**F9:**

SIP/2.0 200 OK  
Via: SIP/2.0/UDP 1.2.3.2  
From: "Alice" <sip:a@example.com>  
To: "Alice" <sip:a@example.com>  
CSeq: 3789 REGISTER  
Call-ID: 8940-3829-194-3492@1.2.3.4  
Contact: <alice@1.2.3.4:5060>;q=0.0;expires=570