

**Czech Technical University in Prague  
Faculty of Electrical Engineering**

# **Doctoral Thesis**

*September 2014*

*Ing. Jiří Hlaváček*

**Czech Technical University in Prague**  
Faculty of Electrical Engineering  
Department of Telecommunication Engineering



# ***ROBUSTNESS OF VOIP SYSTEMS***

**Doctoral Thesis**

***Ing. Jiří Hlaváček***

Prague, September 2014

Ph.D. Programme: Electrical Engineering and Information  
Technology

Branch of study: Telecommunication engineering

**Supervisor: Ing. Robert Bešťák, Ph.D.**

---

## Abstrakt

Telefonní hovor je součástí každodenního života, může zachránit život v případě nouze nebo pomoci překlenout vzdálenost mezi lidmi. Telefonní spojení je tradičně vnímáno jako bezporuchová služba s vysokou dostupností. Internetová telefonie umožňuje implementaci pokročilých služeb a snížení nákladů, ale přináší také komplexnost. Bohužel, systémy internetové telefonie jsou v dnešní době oproti tradičním telefonním systémům stále méně spolehlivé.

Tato práce se zaměřuje na robustnost systémů internetové telefonie (VoIP). V první části je prezentován přehled existujících návrhů a řešení. Pro každé řešení jsou rozvedeny jeho výhody a nevýhody. Z přehledu vyplývá, že existuje kompletní řešení pro síťové zařízení a služby, jakož i pro hardwarové zařízení. Nicméně, optimální architektury programového vybavení jsou stále studovány, zejména řešení pro distribuované a takzvané cloud, tedy na Internetu založené systémy.

Druhá část se zabývá počítačovou virtualizací a možnostmi replikace virtuálních strojů jako platformy pro vysoce dostupné programové vybavení. Nejprve je studován vliv virtualizace na aplikace pracující v reálném čase, jejichž příkladem jsou právě systémy internetové telefonie. Následně je studován mechanismus kontinuální replikace v reálném čase. Z výsledků práce vyplývá, že kontinuální replikace v reálném čase vytváří značné fázové chvění. Toto chvění vylučuje použití tohoto typu replikace pro systémy pracující v reálném čase. V další části je mechanismus migrace upraven tak, aby bylo chvění potlačeno a jeho vlastnosti jsou ověřeny měřeními na laboratorní konfiguraci. Virtualizace spolu s kontinuální replikací v reálném čase je zajímavé řešení pro existující programové vybavení bez podpory vysoké dostupnosti.

Třetí část zkoumá aktorový model (actor model) a jeho aplikace v rámci vysoce dostupných systémů. Nejprve je proveden návrh implementace proxy serveru protokolu SIP s použitím aktorového modelu. Mechanismy umožňující odolnost proti chybám jsou v práci podrobně popsány. Dále je prezentována koncepce modelu dostupnosti vhodného pro programy založené na modelu aktorů. Tento model je hierarchický a umožňuje za pomoci modulů konstrukci komplexnějších modelů. Je založen na barevných stochastických Petriho sítích s logaritnicko-normálním rozložením četností poruch. V závěru této části jsou prezentovány simulace porovnávající dostupnost systému založeného na modelu aktorů se standardním systémem. Tyto ukazují, že aktorový model může zvýšit dostupnost služeb proxy serveru až o dva řády.

Tato práce představuje dvě možné implementace pro vysoce dostupné systémy. Virtualizace spolu s kontinuální replikací v reálném čase je vhodné řešení pro stávající aplikace bez podpory vysoké dostupnosti. Z výsledků práce vyplývá, že při vývoji nové aplikace by naopak mělo být zvaženo použití modelu aktorů.

---

## Abstract

Telephony can save lives in emergency cases or overcome the distance between people. Traditionally, it is known to work with practically no interruption. Voice over Internet Protocol (VoIP) technology brings new blended services and cost efficiency, but also more complexity. Unfortunately, VoIP systems are nowadays still less reliable than the traditional switched systems.

This work deals with the robustness of VoIP systems. In the first part, a survey of existing solutions and propositions is presented and advantages and drawbacks of each possibility are analyzed. It is shown that there are complete solutions for the network hardware and services, as well as for hardware equipments. However, optimal software architectures are still an active research field, in particular for the cloud and distributed environments.

The second part addresses virtualization and its replication capabilities as a platform for high availability software. The impact of virtualization on real-time systems like VoIP servers is measured first and then a continuous live migration mechanism is studied. This work shows that continuous live migration generates an important jitter. The jitter is prohibitive for use of this type of migration in the real-time systems. The migration mechanism is improved and its behavior is validated by a measurement on a testbed. A virtualized server with the improved continuous live migration is an interesting solution for existing software without high availability support.

The third part studies the actor model and its application to highly available systems. A high availability implementation of a SIP proxy is proposed. Mechanisms enabling fault tolerance are explained in detail. A modular availability model suitable for actor based applications is conceived. The model is hierarchical and scales well. It is based on stochastic color Petri nets with lognormal distribution of failure rates. Simulations were performed to compare the availability of the actor based system with a standard one. Results show that the actor model can improve the availability by a magnitude of two orders.

The work presents two possible implementations of a high availability system. A virtualization based one, which is suitable for existing applications without high availability support. And an actor model based one, which should be considered when developing a new application.

---

## Acknowledgements

First and foremost I would like to express my thanks to my supervisor, Dr. Robert Bešťák. He was patient and available all the time. Our discussions always brought a new point of view and enriched my work. His remarks helped to improve and clarify the result.

I would also like to thank Mrs. Renata Kroutilíková for her help with the administration while I was in France.

My thanks are also due to all the participants of the MOTELI project, in particular to Stephane Bayer and Jean-Yves Lebleu, I had the chance to learn a lot from them. A special thanks goes to William Caldwell for its valuable remarks and reviews.

And finally I would like to all my family for their infinite patience and continuous support.

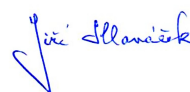
---

## Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with methodical instructions about ethical principles for writing academic thesis.

Prague, September 2014

Jiří Hlaváček



---

## Table of Contents

<b>Abstrakt.....</b>	<b>3</b>
<b>Abstract.....</b>	<b>4</b>
<b>Acknowledgements.....</b>	<b>5</b>
<b>Declaration.....</b>	<b>6</b>
<b>List of Figures.....</b>	<b>10</b>
<b>List of Tables.....</b>	<b>11</b>
<b>Acronyms.....</b>	<b>12</b>
<b>1 Introduction.....</b>	<b>14</b>
1.1 Architecture of VoIP Systems.....	14
1.2 Background and Motivation.....	14
1.3 Context Precision.....	15
1.4 Problem Statement.....	15
1.5 Aims of Thesis.....	15
1.6 Thesis Outline.....	16
<b>2 VoIP Systems Principles.....</b>	<b>17</b>
2.1 VoIP Evolution.....	17
2.2 Architecture of VoIP Systems.....	17
2.2.1 Signaling.....	17
2.2.2 Media.....	17
2.3 SIP Protocol.....	18
2.3.1 Basic Principles.....	18
2.3.2 Layers.....	18
2.3.3 Components.....	19
2.3.3.1 Registrar.....	19
2.3.3.2 Location Service.....	19
2.3.3.3 Proxy.....	20
<b>3 Current Situation.....</b>	<b>21</b>
3.1 Comparison of VoIP and Legacy Systems.....	21
3.2 Availability of Terminals.....	22
3.3 Redundant Network Architecture.....	22

---

3.3.1 SIP High Availability Network Architecture.....	23
3.3.2 Failover of Routing Protocol.....	26
3.3.3 Redundancy of Network Services.....	26
3.4 Availability of VoIP Servers.....	27
3.4.1 Fault Recovery without Context Preservation.....	28
3.4.2 Fault Tolerance with Context Replication.....	29
3.4.2.1 Solutions Impacting Applications.....	29
3.4.2.2 Solutions Transparent for Applications.....	30
3.4.3 Software Robustness.....	31
3.5 Conclusion.....	31
<b>4 Virtualization.....</b>	<b>32</b>
4.1 Related Work.....	32
4.2 Problem Analysis.....	33
4.2.1 Virtualization Basics.....	33
4.2.2 Continuous Live Migration.....	34
4.2.2.1 Data Consistency.....	34
4.2.3 Proposed Solution.....	35
4.2.3.1 Principle.....	35
4.2.3.2 Implementation.....	35
4.2.4 Obtained Results.....	36
4.2.4.1 Testbed Description.....	36
4.2.4.2 Jitter Calculation.....	36
4.2.4.3 Virtualization Impact.....	37
4.2.4.4 Live Replication Impact.....	37
4.3 Conclusion and Future Work.....	40
<b>5 Actors.....</b>	<b>42</b>
5.1 Introduction.....	42
5.2 Related Work.....	43
5.3 Introduction to Akka.....	44
5.3.1 Actor Model.....	45
5.3.2 Actor's Life-cycle.....	45
5.3.3 Supervision, Monitoring and Fault Tolerance.....	46
5.3.4 Finite State Machines.....	46
5.4 SIP Server Implementation.....	47



---

5.4.1 Component Model.....	47
5.4.2 Actor Model.....	47
5.4.2.1 Transport Module.....	47
5.4.2.2 Registrar and Location Module.....	47
5.4.2.3 Proxy Module.....	49
5.4.3 Fault Handling.....	49
5.4.3.1 Registrar Router and Location Manager.....	49
5.4.3.2 Session Router.....	49
5.4.3.3 Transport Supervisor.....	49
5.4.4 Timing Constraints.....	50
5.5 Availability Model.....	50
5.5.1 Single Actor Model.....	50
5.5.2 SIP Transport Model.....	52
5.6 Availability Analysis.....	53
5.7 Numerical Results.....	54
5.8 Conclusion.....	57
<b>6 Contributions of the Thesis.....</b>	<b>58</b>
<b>7 Conclusions.....</b>	<b>59</b>
<b>8 Future Work.....</b>	<b>61</b>
<b>9 Publications.....</b>	<b>62</b>
9.1 Publications Related to the Thesis.....	62
9.1.1 Publications in Impact Journals.....	62
9.1.2 Publications in Reviewed Journals.....	62
9.1.3 Publications Excerpt in Web of Science.....	62
9.1.4 Other Publications.....	62
9.2 Unrelated Publications.....	63
9.2.1 Other Publications.....	63
<b>10 References.....</b>	<b>65</b>

---

## List of Figures

Figure 1: Session Initiation Protocol layers.....	19
Figure 2: SIP components.....	20
Figure 3: Redundant network architecture.....	22
Figure 4: SIP high availability network design.....	23
Figure 5: SIP high availability network with Service Access Points.....	24
Figure 6: Network buffering during continuous live migration.....	34
Figure 7: Testbed configuration.....	37
Figure 8: Jitter without virtualization.....	38
Figure 9: Jitter measured calling via a virtualized Freeswitch server.....	38
Figure 10: Jitter using unmodified replication mechanism.....	39
Figure 11: Jitter using modified replication mechanism.....	39
Figure 12: Jitter and packet loss during the migration following a failure.....	40
Figure 13: Resolved (green) and unresolved (red) bugs in Asterisk IPBX.....	42
Figure 14: Actor system tree.....	45
Figure 15: Actor life-cycle.....	46
Figure 16: SIP server component model.....	48
Figure 17: Single actor model.....	51
Figure 18: Single actor module interface.....	52
Figure 19: Transport layer availability model.....	52
Figure 20: Two-state Markov chain.....	53
Figure 21: Hazard rate of the lognormal distribution.....	55
Figure 22: Availability model of considered standard implementation.....	56

---

## List of Tables

Table 1: Availability comparison between an actor based system and a standard one.....	56
Table 2: Impact of the failure rate on the availability.....	57

---

## Acronyms

3GPP	3 <sup>rd</sup> Generation Partnership Project
AMF	Availability Management Forum
ASCII	American Standard Code for Information Exchange
BFD	Bidirectional Forwarding Protocol
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DN-LB	Domain Based Load Balancer
DNS	Domain Name System
DNS SRV	Domain Name Server Service Record
DPD	Death Peer Detection
EWSD	Electronic World Switch Digital
FFF	Fast Failure detection and Failover
FSM	Finite State Machine
GB	Gigabyte
HSRP	Hot Standby Router Protocol
IETF	Internet Engineering Task Force
IETF	Internet Engineering Task Force
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IPBX	Internet Protocol Private Branch Exchange
IT	Information technology
ITU	International Telecommunication Union
LAN	Local Area Network
LVS	Linux Virtual Server
MPLS	Multi-Protocol Layer Switching
MTBF	Mean Time Between Failures
NGN	Next Generation Networks
OpenAIS	Open Application Interface Specification
OS	Operating System
OSI	Open Systems Interconnect

---

PBX	Private Branch Exchange
PC	Personal Computer
RAM	Random Access Memory
RFC	Request For Comments
RTP	Real-time Transport Protocol
SAP	Service Access Point
SCTP	Stream Control Transmission Protocol
SIP	Session Initiation Protocol
SMP	Symmetric Multiprocessing
SPOF	Single point of Failure
SRTP	Secure Real-Time Protocol
TCP	Transmission Control Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UDP	User Datagram Protocol
URI	Universal Resource Identifier
VM	Virtual Machine
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network
VRRP	Virtual Router Redundancy Protocol

# 1 Introduction

VoIP systems rely on the existing computer network and propose advanced multimodal services such as unified communications. Unified communications systems integrate different exchange channels into one centralized system. Each user can choose the best way to communicate with other users based on the topic's urgency, the presence information, the surrounding environment and other factors. Unified communication systems can provide services like messaging similar to mail or chat, audio or video calls, audio or video conferences, screen sharing. Telephony services are traditionally highly available, five nines availability is the standard achieved thanks to a specific hardware and software equipments. VoIP systems are popular because of cost efficiency reached using standard hardware and software, but there's an important impact on robustness. The second factor limiting the availability of VoIP systems is the integration with a shared network infrastructure and the third is the interaction with other information technology systems such as directory services. Nowadays, private branch exchanges (PBX) providing unified communications services are often complex distributed systems serving users on many distant sites interconnected by a virtual private network (VPN). Integration with other information technology systems brings the possibility to implement many advanced services, for example, a web callback.

## 1.1 Architecture of VoIP Systems

VoIP systems are built on top of the IP network. Main components are VoIP servers and IP phones. IP network ensures end-to-end data connectivity and transport. VoIP specific services are ensured by specific protocols at the 7th layer of the Open System Interconnect (OSI) reference model [1].

## 1.2 Background and Motivation

Contrary to the traditional PBX, which is often deployed as a standalone system, Internet protocol PBX (IPBX) is deployed in a complex environment with many interconnections. New challenges like the cloud and distributed environments are faced. Operators offer five nines availability for legacy systems but usually about three nines for VoIP systems, e. g. 99.95% by French Orange Business [2]. An IPBX's environment is composed of a network infrastructure, hardware and software. Network availability can be ensured using

redundant paths and appropriate configuration protocols. A high availability hardware with redundant components is also widely available. A typical mean time between failures (MTBF) of a server is about 6.5 year [3], but usual OS's MTBF is about 60 days [4]. IPBX software enables advanced services and integrations, but its availability is one of the VoIP system's drawbacks.

### **1.3 Context Precision**

The problem is less significant in the core network such as IMS. The IMS standard describes a robust sophisticated infrastructure, separation of control, data and management planes, whereas an IPBX is often represented by a single server or by a cluster of two servers ensuring hardware availability. A software crash will impact all IPBX's users. Also, IPBXs are often incorporated into the information technology (IT) infrastructure and provided services interact with third party servers.

### **1.4 Problem Statement**

An IPBX providing advanced services and integration with other services requires more complex software. New software architectures are needed to be able to provide the same availability while increasing software complexity. Software architecture should be fault tolerant and allow context replication or restore on failure in order to preserve service continuity. The software should be able to overcome hardware malfunctions. It should also enable reliability modeling and estimation of the availability characteristics like the MTBF. Two active research fields are identified. The first one is focused on software architectures in the cloud computing and the second on the distributed software applications [5] [6] [7].

### **1.5 Aims of Thesis**

With consideration of the actual situation of the studied problem, the aims of this thesis were defined as follows:

- Analyze drawbacks of existing solutions and architectures for high availability VoIP systems with a focus on private branch exchanges. An important axis of the analysis is the way the implementation is dealing with software failures.
- Find software architecture suitable for implementation of highly available VoIP systems. The emphasis is on the call and service continuity, the call context must, therefore, be restored or replicated following a failure. The architecture should be applicable to the distributed and cloud environments.

- Propositions should be validated on a prototype in order to validate its feasibility.
- The availability should be validated either by measurements, or by simulation.

## **1.6 Thesis Outline**

This work adopts important parts from author's published articles listed in the Referenced as [8], [9] and [10]. In the following section, I present principles of VoIP systems. A survey of the existing solutions for highly available platforms is provided in the third chapter. The fourth chapter focuses on computer virtualization and its features for the fault tolerance. The actor model and its applications are discussed in the fifth chapter. Chapters 6, 7 and 8 present thesis contributions, conclusions and future work topics respectively.



## 2 VoIP Systems Principles

### 2.1 VoIP Evolution

Voice over Internet Protocol (VoIP) technology allows use of the Internet Protocol (IP) networks as a support for telecommunication systems. This technology was practically introduced by the International Telecommunication Union (ITU) in its recommendation describing H.323 protocol [11] by the end of 1996. This recommendation specifies a first version of new protocol for multimedia communication between terminals over Local Area Networks (LAN). In 1999, the Internet Engineering Task Force (IETF) released a first version of the Session Initiation Protocol (SIP) [12]. The first commercial use of the VoIP technology is dated back to 1995. A real expansion of VoIP networks started in 2004. The IP protocol became practically the standard for cheap but powerful worldwide transmission networks. Together with MultiProtocol Label Switching (MPLS) mechanism it gave birth to the Next Generation Networks (NGN). Telecommunication operators are nowadays more and more using IP based NGN and networks with an IP Multimedia Subsystem (IMS) core.

### 2.2 Architecture of VoIP Systems

Main components of VoIP systems are an IP network, VoIP servers and IP phones. VoIP servers process call signaling and can also process user data flow. IP network must reach VoIP requirements such as low packet loss and low latency and jitter.

#### 2.2.1 Signaling

There are two main VoIP signaling protocols. H.323, that is more bandwidth efficient, and SIP, that is less complex and more simply extensible than H.323 [13]. SIP protocol was chosen for VoIP signaling in the IMS architecture and in mobile networks by the 3GPP (3rd Generation Partnership Project). Thanks to these facts SIP is nowadays more used than H.323. The rest of this work is therefore focused on SIP.

#### 2.2.2 Media

Besides the signaling, the VoIP systems transport media, e.g. voice and video. It's a real-time data flow where the low latency is more important than packet loss. The most often

employed is the Real-time Transport Protocol (RTP) [14]. The Secure Real-time Transport Protocol (SRTP) [15] or other real-time transport protocol can be used.

## **2.3 SIP Protocol**

SIP is an ASCII based protocol designed to manage establishment of sessions. It's easily extendible; presence management and messaging are two examples of add-ons. In this paragraph, I focus on principles compulsory for comprehension of this work, a complete description of the protocol structure is detailed in the section 5 of the protocol specification [16].

### **2.3.1 Basic Principles**

SIP users are identified by a SIP Uniform Resource Identifier (URI). Its format is similar to the email address; a simple SIP URI is:

- `sip:user@voip.server.com:port`

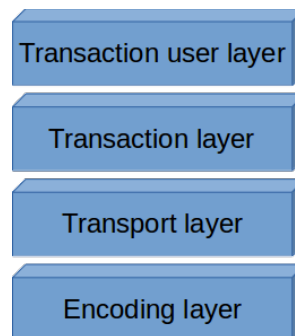
User's phone registers to the server with the user's URI. The server stores the user's location for future request routing. When a user makes a call, its phone issues a request to the server, the server looks up the address of the called user's phone and routes the request to the next hop. The next hop can be the caller's phone, if the called user is in the same domain as the caller, or another server.

SIP is a layered protocol. Encoding, routing and other services are mapped to layers for easier orientation.

### **2.3.2 Layers**

The lowest layer is the encoding layer implementing an augmented Backus-Naur Form grammar defined in the protocol's specification. Above the encoding layer is the transport layer. This layer adapts messages to the transport protocol. SIP works with three transport protocols: user datagram protocol (UDP) [17], transmission control protocol (TCP) [18] or stream control transmission protocol (SCTP) [19]. Messages coming from the encoding layer are processed by the transport layer and sent to the transaction layer if this layer is present. The transaction layer is a set of state-machines processing SIP messages. These state-machines are completed by message processing rules. A transaction provides a context for treatment of dependent messages. A successfully terminated transaction can establish a dialog: a time-limited object serving as a context between dependent transactions.

State-machines in the transaction layer communicate with the transport layer, but also with the transaction user layer. The latter implements SIP services by controlling the transactions. As an example I can detail a call establishment. An Invite request from an endpoint A is received by the encoding layer, processed by the transport layer and a transaction is created in the transaction layer. The transaction user layer creates a new transaction, which initiates a new Invite request to the endpoint B. This request is passed to the transport and encoding layer and sent to the endpoint. The Fig. 1 depicts SIP protocol layers.



*Figure 1: Session Initiation Protocol layers*

### 2.3.3 Components

Layers describe the structure of the SIP protocol. Components providing different services can be defined using these layers. According to the specification, main SIP server components are registrar, location service and proxy. Services and structure of these components is reviewed in this section. Interactions between components are shown in Fig. 2.

#### 2.3.3.1 Registrar

A registrar processes Register requests and saves bindings of address-of-record keys to zero or more contact addresses to the location service. The registrar processes also Register refresh request. The registrar is implemented in the transaction user layer and uses all the inferior layers. More details can be found in the section 10 of the SIP protocol specification [16].

#### 2.3.3.2 Location Service

A location service stores registration bindings and provides them to the proxy when requested. However, the location service takes care of registration lifetime. The location service is a companion of the Registrar and a service for the proxy.

### 2.3.3.3 Proxy

The primary proxy's goal is routing of SIP messages using the bindings from the location service. It can also act as a policy enforcement server or can be extended to allow advanced services like 3rd party call control. The proxy can be stateless or stateful, depending on whether the proxy implements the transaction layer. In this work, I focus on the stateful proxy, because stateless proxies can't implement any service requiring the call context. A stateful proxy is, like the registrar, implemented in the transaction user layer and uses all the inferior layers. A detailed description of its function is described in the section 16. of the SIP protocol specification [16].

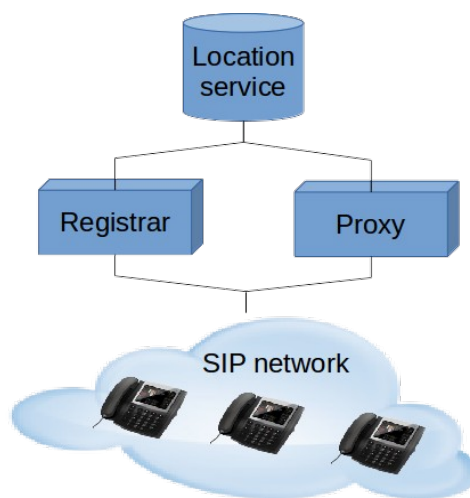


Figure 2: SIP components

### 3 Current Situation

The availability is an important issue of VoIP systems. VoIP technology was developed as an alternative to the legacy systems with objectives to be cheaper and to provide more services. VoIP servers are often running on standard servers with common operating systems like Linux. It's difficult to reach the availability of the legacy systems which are running specific fault-tolerant software on servers with redundant components. Weaknesses of the VoIP systems in terms of availability are analyzed in this chapter. Known solutions and related work are also presented.

Basic principle of high availability systems is the absence of single points of failure (SPOF). The single point of failure is an element required for the faultless operation of the systems. It can be eliminated by a backup unit. A faulty unit can also be restarted. In VoIP systems there are two types of elements: stateless and stateful elements. Stateless elements are easier to supersede as only the configuration must be the same. Stateful elements maintain a state, in case of failure the state must be replicated to the backup element or restored once the faulty element restarted. Classification to stateful and stateless elements is employed in next paragraphs.

#### **3.1 Comparison of VoIP and Legacy Systems**

Analog and digital circuit switched communication systems propose at least 99,999 % availability, i. e. five minutes of downtime per year. This is achieved by using specific redundant hardware and fault tolerant software together with redundant transmission lines and a continuous repair service. IP networks used as a transit network in VoIP systems were not initially projected for transport of real-time data. Most IP routing failover mechanisms not were conceived for real-time data transmissions in mind. Failover time of routing protocols can reach a minute; data transmission service type is the best effort. VoIP systems require a failover in order of hundreds of milliseconds (for safety-critical systems) and a guaranteed QoS. Safety critical VoIP systems exist already, but these are made to measure and are not publicly available.

### 3.2 Availability of Terminals

The Mean Time Between Failures (MTBF) of today's terminals is about 200 000-400 000 hours depending on the manufacturer, i. e. more than 20 years. Likely the terminal is replaced by a new one before its fail. Some VoIP terminals offer a possibility to be connected to two different segments of VoIP network, which increases the overall availability. Another high-availability feature often integrated in the phone is to configure a backup register server. It can be used if the DNS server is not used during the registration process.

### 3.3 Redundant Network Architecture

Highly available VoIP networks are usually using a specific network architecture defined for high availability servers. This architecture is based on redundant interconnections combined with redundant network devices and servers. An example of such architecture is shown in Fig. 3. SIP proxies and terminals must be equipped with two network cards. Furthermore, the Internet Protocol (IP) failover mechanism [20] must be implemented. The principle of this mechanism is that the active server uses a virtual IP address which is taken by the backup server when the active one fails. Thanks to that, the process is entirely transparent to the clients.

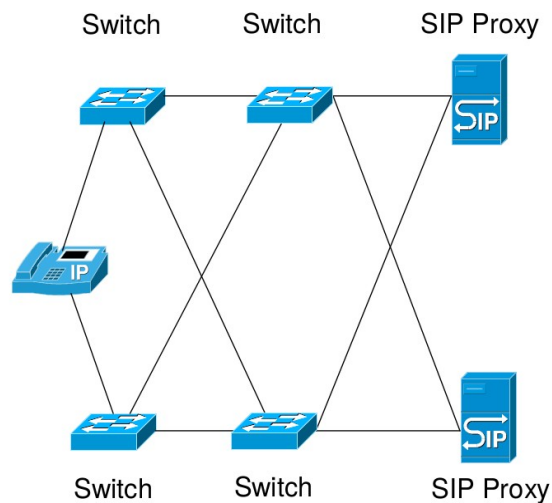


Figure 3: Redundant network architecture

A detailed description of the network architecture with IP failover can be found in [21]. This architecture must be used together with specific switching and routing protocol in order to detect failures as fast as possible. These protocols are discussed below.

Article [22] describes an interesting way of availability evaluation using a matrix representation of network devices and links. This method offers a simple graphical representation of a system, which can be used to find out network weaknesses and to have a consolidated view of network configuration. It can also help with elaboration of availability metrics and equations.

### 3.3.1 SIP High Availability Network Architecture

The architecture presented in the previous paragraph shows the basic idea, but doesn't scale well. As the number of system users increase, more proxies can be required. For security and performance reasons, SIP proxies are often behind a SIP dispatcher. The SIP dispatcher is a call stateless SIP proxy, however it must dispatch requests and responses belonging to one session to the same SIP proxy. It can be done in a stateful way – e. g. by a hash-table, or in a stateless way – e. g. users with numbers in the first range are served by the first proxy, users with numbers in the second range are served by the second proxy etc. The basic idea is developed in order to eliminate the SPOF on a border of SIP network represented by the dispatcher. The SIP high availability architecture use two or more dispatchers and two or more SIP proxy/registrar servers.

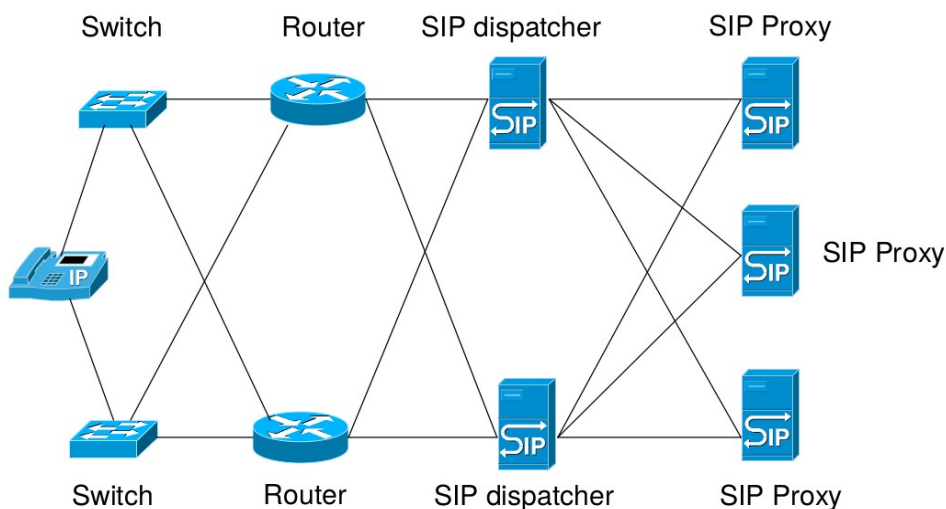


Figure 4: SIP high availability network design

The availability is ensured by a migration of IP address in case of dispatcher failure and by a monitoring of SIP proxies state. Increasing the number of SIP servers will increase the availability of the VoIP network. The described architecture is shown in Fig. 4.

Some implementations of SIP high availability network architecture are already industrialized. A standardized solution is based on the Virtual Router Redundancy Protocol (VRRP) [23]. This protocol enables a pair of machines on a LAN to negotiate an ownership of a virtual IP address. One server is active and use the virtual IP address, other one is in standby mode and takes over if the active one fails. This protocol doesn't scale well; it works only for two servers (on active and one standby). Cisco defined a proprietary protocol that proposes the same functionality called Hot-Standby Routing Protocol (HSRP) [24] which scales better, the maximal number of redundant routers isn't limited by the protocol. The drawback is that this protocol is proprietary.

Researchers from Taiwan presented in [25] a solution based on the Service Availability Forum service specification called OpenAIS (Open application interface specification) [26]. They named their scheme Fast Failure detection and Failover (FFF). Presented system architecture is adopted from the SIP high availability architecture.

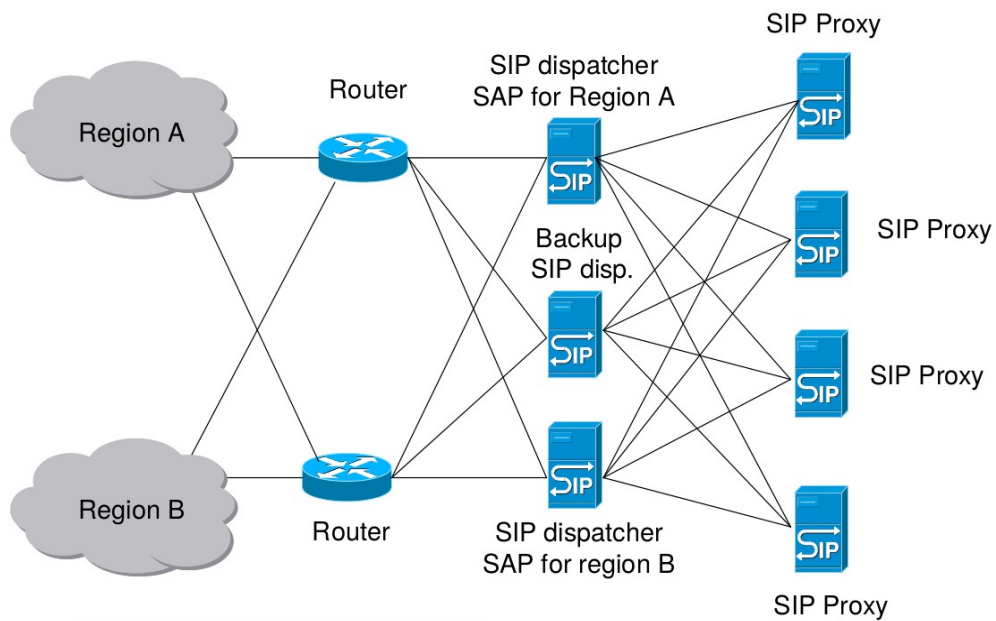


Figure 5: SIP high availability network with Service Access Points

A new component called Service Access Point (SAP) is introduced. SAP is a virtual IP address used by one of SIP dispatchers. Service Access Points hide redundancy and



facilitate network configuration and scalability. Proposed design is shown in Fig. 5 (inspired by [26]).

The principle is the same as in the previously cited solution: a virtual IP address is used by active and operational SIP server (dispatcher or proxy) and in case of failure is transferred to the backup one. The difference is in the way a failure is detected and a new active server is elected. The VRRP protocol is replaced by the OpenAIS Availability Management Framework (AMF). The proposition was implemented and tested using a standard PC. Tests were executed under a different CPU load. The Fast Failure detection and Failover scheme reduces the failover time by 80% compared to the VRRP solution. The failover time is then about 0.6 second using FFF compared to 3.1 second using VRRP. It is a valuable improvement. The main inconvenient of this solution is the need of some custom software development.

The paper [27] presents another way implementation of the SIP dispatcher. The proposition is based on Linux technology called IP virtual server (IPVS). An implementation called Linux Virtual Server (LVS) is used to implement a proof of concept. LVS is a framework for load balancing and high availability clustering. The switching architecture is based on three components: servers, routing mechanism and scheduling mechanism. Load balancers are called Directors and SIP proxies are called Real servers. The Director server implements a request scheduling mechanism and together with other network components also a routing mechanism. The Real server handles client requests. The LVS's IP address is the only address known by SIP clients. Scheduling of SIP requests is done using a distribution function. This function is based on the server capacity which is transformed in weight. It is ensured that all SIP requests that have the same Call-ID are sent to the same Real server (SIP proxy). This is necessary for stateful processing of requests. Described mechanism allows use of different SIP proxy implementations from different sources, which is useful for high availability systems. Another important point proposed in this paper is the quality of system maintenance. The maintenance of high availability system needs to be effective and completely controlled. The proposed system is able to isolate one or more servers for maintenance and to introduce a server gradually: server should be tested by a few real requests before serving the planned load. This allows administrator to verify the configuration and stability following a reparation or new deployment. The main drawback of this solution is the complexity of the Linux Virtual Server architecture.

### 3.3.2 Failover of Routing Protocol

One of main issues of SIP high availability network is a convergence of routing tables. The convergence time is conditioned by a detection of link failure. The paper [28] explains a death-peer-detection (DPD) mechanism based on a fuzzy logic. This solution is flexible and can be used independently of routing protocol. Unfortunately, the algorithm needs a long training period. Another important inconvenient is that each change of system configuration must be followed by a new training period.

Article [29] describes a solution that allows decreasing the time of link failure detection and update of routes. Propositions are based on protocol independent Hello mechanism called Bidirectional Forwarding Protocol (BFD). This protocol is standardized by the IETF [30]. BFD is designed to detect failures of bidirectional links between two forwarding engines with emphasis on very low latency. Although this protocol can be used by the 2nd or 3rd layer of OSI model, manufacturers implements it only on 3rd layer. A three-way handshake is used for initialize the state of each neighbor; possible states are down, init (for initialization) and up. Contrary to common routing protocols like OSPF, the interval of BFD's Hello messages can be configured in a range of few milliseconds. The neighbor state passes down after a specified number of BFD packets are lost. It is shown that the minimal period of Hello packets is between 5-50 ms, otherwise there's a danger of erroneous detection of failures and occurrence of oscillations in routing tables. The conclusion is positive; a failover time about 200 ms can be achieved using a BFD protocol on a standard COTS router (Cisco or Quagga software on a Linux machine). However, this result is conditioned by a customized configuration of network devices.

### 3.3.3 Redundancy of Network Services

Support functions like DNS and DHCP propose standardized failover mechanisms that minimize the rupture of service. However, the coherency between DHCP and DNS databases must be ensured by the configuration. This doesn't reflect needs of VoIP systems; the coherency needs to be ensured automatically in order to simplify the configuration, protect from configuration errors and minimize the possibility of incoherences. Standardized failover mechanisms are based on master/slave architecture. As every SIP terminal will always ask the master DNS server first, this architecture adds a delay to the call establishment and it also complicates the configuration and maintenance. A specific configuration, which ensures the coherency between DHCP and DNS databases, is proposed in [21]. This solution connects DHCP and DNS tables to a SQL database. Used SQL database must be able to replicate data from one server to another. A database

replication algorithm ensures the coherency of data between redundant servers. Changes of DHCP's and DNS's configurations are done through the database. A user interface is interfaced with the database and therefore it is independent from DHCP and DNS. Described solution allows both DHCP and DNS servers to run in master-master mode and therefore it is easier to change failed server. It is based on standard DHCP and DNS daemons, SQL database and a small script that updates DHCP and DNS tables.

### **3.4 Availability of VoIP Servers**

SIP servers are critical components of SIP networks. SIP proxies are relaying SIP messages between terminals; each server therefore represents a single point of failure. SIP servers are software applications running on standard server hardware.

Modules used in circuit switched systems have a MTBF about 1 000 000 hours (approx. 100 years, based on MTBF of German Electronic World Switch Digital (EWSD) modules). Although today's servers may have multiple processors, redundant network cards, several hard-disks working in a RAID and redundant power sources; their MTBF doesn't reach MTBF of circuit switched system specific hardware (MTBF of servers is usually between 50 000 and 300 000 hours). Exceptions are routers and specialized router cards with MTBF about 1 000 000 hours (Cisco router modules). Large legacy systems like the German EWSD system implement software recovery controllers and hardware monitors [31]. Software recovery controllers are special programs, which aim to detect, solve and report errors. Hardware monitors test periodically each component of the system. Each anomaly is logged up and reported. As far as I know, software implementations of SIP servers aren't that robust, interfaces for software recovery controllers aren't proposed. Hardware monitoring of servers should be done by some specialized hardware monitor with problem reporting. Actual VoIP platforms are represented by a standard redundant hardware together with an operating system configured to provide high availability services. Software architectures of VoIP servers usually do not implement recovery and fault tolerance, therefore they are not as robust as the software of the legacy systems. The Asterisk PBX is a multi-threaded C application [32], therefore sensitive to deadlock and with possible segmentations faults. The Mobicents suite runs in the JBoss application server [33] [34], its error-recovery features are also limited.

A document from Cisco [35] compares the availability of the legacy PBX to the Cisco VoIP systems. It is shown that both approaches can achieve five nines availability

(5.26 minutes of downtime per year) by replication of components. This is a special case, because Cisco SIP servers are not running on a standardized server but directly by the Cisco router. Also, the features of the application programming interface (API) exposed by the presented Cisco system are very limited. A generic VoIP system can use switches and routers with the MTBF values close to routers or circuit switched systems. Anyhow, the server running a SIP application doesn't reach the MTBF value of the router. A server redundancy must be introduced in order to achieve five nines availability using standard servers.

Based on whether the call and associated services are lost or preserved following a failure, two types of recovery exist: fault recovery without context preservation and fault tolerance with context replication.

### **3.4.1 Fault Recovery without Context Preservation**

The simplest implementation of the server redundancy is to introduce an address of a backup proxy/registrar server. This solution is only static and doesn't scale well. It's useful only as a cheap solution for small local networks. Also, if a registrar/proxy fails, established calls are lost and the endpoint is not reachable before it re-registers with the new active registrar/proxy server. One of the recommended methods to ensure service continuity in case of server failure is the use of the Domain Name System Service Record (DNS SRV) mechanism [36] [37]. This solution discussed in [38] is called "Intelligence in previous hop". The idea is quite simple: a special DNS request is issued to get an IP address of the SIP proxy/registrar. This implies that SIP components must implement DNS SRV lookups. The abbreviation DNS SRV means an IP address of the server that proposes the required service. The response from the DNS server is a DNS SRV record, which is a list of IP addresses; each address is associated a weight and a priority. This solution is standardized for the SIP core network, but most terminals doesn't implement DNS SRV mechanism. Furthermore, the use of DNS requests increases call establishment time, in particular in the case of primary DNS failure. In this method, a single point of failure is the DNS server itself. A secondary DNS server is usually present but the timeout and takeover adds some seconds to the call establishment. Also, the context replication issue is not addressed and thus this solution is suitable only for stateless servers.

The paper [39] explains a solution that allows using of standard DNS requests to redirect SIP terminals to one of multiple SIP servers. The idea is to introduce a domain based load balancer (DN-LB). The load balancer uses a heart-beat probe mechanism to check availability of SIP servers and in function of result redirects SIP clients to one of the

pool of SIP servers. SIP clients must address DNS lookup requests to the domain based load balancer. This solution is quite easy to implement using existing software products. The main inconvenient of this solution is that the domain based load balancer represents a single point of failure. This difficulty can't be solved by using two domain based load balancers, because the DNS clients wait for some seconds before sending the request to the second DNS server (DN-LB here). This is hardly acceptable in a highly available system. Another weak point is that SIP terminals are required to do a DNS lookup before sending a SIP request. This feature is not proposed by standard SIP terminals.

It's possible to let the media flow pass directly between terminals, without passing by the proxy. Such a configuration allows maintaining call even when a proxy failure occurs, but as the call context is lost, additional services can't be offered.

### **3.4.2 Fault Tolerance with Context Replication**

A common point of the solutions described below is the use of the Internet Protocol (IP) failover mechanism discussed in the section 3.3.1.

The passage of the IP address from failed server to a backup one is fundamental for the VoIP service continuity. However, ensuring a reliable delivery of SIP messages to the operating SIP server isn't enough to ensure a successful failover without lost calls. The new active server doesn't know current calls nor registered terminals. Consequently, as the new active server doesn't know any endpoints, any incoming calls can't be processed before a new terminal registration. SIP allows use of stateless proxies, but operators need call stateful proxies for billing etc. New active server must therefore restore contexts of the established calls and registered terminals in order to ensure the continuity of services. Call establishment, call transfers, charging and other complementary services then continue to be provided by the new active server. This mechanism can work only when UDP [19] is used. TCP [19] is connection oriented and when the connection fails the terminal must open a new connection with the server. If this condition is fulfilled, described mechanism can be used even using TCP or TLS [40]. A way to ensure failover of the TCP connection is standardized in [41].

The call context is saved by the application. There are two types of fault tolerance: solutions affecting applications and solutions transparent for applications.

#### 3.4.2.1 Solutions Impacting Applications

An application level replication is described by A. Gorti in [42]. The proposed solution requires a specific software development, which is expensive and long. Replication enabled applications based on software frameworks for high availability (e.g., Terracota)

are hard to configure and maintain. Furthermore, there are several requirements on the software architecture such as thread safety [43]. G. Kambourakis et al. propose a database-based state sharing mechanism [44]. Contexts are saved in a database and the replication is done by the database engine. This solution is relatively easy to implement as only a database connector needs to be developed instead of a complex replicated system preserving data consistency. Nevertheless, the architecture remains complex and hard to integrate with existing applications.

The paper [45] describes a solution using OpenAIS AMF for call context replication. It doesn't consider the replication of registrations. Call contexts are replicated using checkpoints. A checkpoint is defined as a current call state. The aim of checkpoints is to simplify context replication. In order to define appropriate number of checkpoints a functional analysis of SIP protocol should be done. The article doesn't explain how checkpoints were chosen. The focus is on the implementation and its complexity. Following areas of application design were identified as keys to the high availability application simplicity: event-driven model, checkpoint data identification and process roles. It was shown that using discussed solution the availability of a VoIP service can be improved from 99,999 % to 99, 9999 % without considering hardware faults. The presented solution can be considered as a proof of concept, because its real performances are very low.

The paper [46] uses the OpenAIS framework to supervise SIP proxies by the dispatcher. It proposes an OpenAIS-based SIP load balancing strategy. Proxy health conditions and CPU load are monitored by the dispatchers using OpenAIS functions. This solution brings down the time needed to find out a failed proxy and improves the balancing thanks to the knowledge of the charge of each proxy. The main disadvantage of this solution is the need of some custom software development.

A common problem to all application dependent solutions is the breakdown of TCP and Transport Layer Security (TLS) connections when taking over the IP address. These connections can't be migrated without specialized operating systems or replication aware clients.

#### 3.4.2.2 Solutions Transparent for Applications

Solutions transparent for application usually require more resources, but it is compensated by an easier development and maintenance of applications.

For special cases of Java applications, Aspect-oriented programming frameworks with real-time byte-code instrumentation for context replication can be used [47]. The cited

work discussese drawbacks of this solution. Main issues are complexity of use and performances.

The computer virtualization is also an interesting alternative discussed in chapter 4.

### **3.4.3 Software Robustness**

The most of VoIP software implementations do not include fault tolerance nor rejuvenation. These techniques help to achieve the required availability in highly available systems. Works [48] and [49] show that software implementations capable of rejuvenation can improve the availability of VoIP systems.

## **3.5 Conclusion**

It is shown that VoIP software solutions are not mature enough to meet the availability of circuit switched telecommunication equipments without using or developing expensive proprietary solutions. However, results of research work demonstrate that it's possible to ensure at least five nine availability by combination of VoIP technology with some standardized high availability technology completed by some specific software development. Issues of proposed solutions are high failover time and complicated and therefore costly implementation. Use of proprietary solutions brings problems with interoperability. Furthermore, discussed propositions don't reflect security and maintenance requirements. Another problem that is never addressed is the failover mechanism of TCP or even TLS SIP links.

## 4 Virtualization

Replication enabled applications are complex and therefore expensive. Thus, other solutions are under research.

Virtualization is widely used for its advantages such as better resource usage, flexibility and scalability. Moreover, it can also be employed to improve system availability using live migration techniques. The main advantage of virtualization consists in the fact that the migration is completely transparent for applications. Live migration mechanisms are still in development. The main challenges include scheduling of virtual machines [50] and network bandwidth optimization [51]. Other issues are latency, jitter and packet bursts introduced by the replication process itself.

In this section, I propose a modification of the network buffer used by the replication mechanism. This buffer enables saving of all outbound data between two successive replication steps to ensure data consistency in case of failure. The proposition consists in network packet inspection and classification, where only Transmission Control Protocol (TCP) packets are buffered. User Datagram Protocol (UDP) and Real-time Transport Protocol (RTP) packets are forwarded, bypassing the live replication buffering. This way, the latency and jitter for RTP packets are minimized and the voice quality of calls is improved. The proposed modification minimizes virtualization's impacts on real-time data flows.

### 4.1 Related Work

Virtualization techniques are nowadays very popular, mainly because of their better hardware resources usage. These techniques are also employed to facilitate hardware maintenance and fault tolerance. A special use case is continuous live migration of running Virtual Machine (VM) used for high availability systems. The principle of the continuous live migration is to replicate the primary machine state to the backup machine continuously and to start the backup machine if the primary one fails. However, this technique is resource demanding. It also introduces a periodic short interruption necessary for synchronization to the primary machine execution. Two ways of machine's replication exist. The first way is based on replay of non-deterministic events received by the primary machine on the backup one [52]. This technique is used for example by VMware [53]. The deterministic replay is not adapted to the Symmetric Multi Processing (SMP) environments



since ordered memory access is needed [54]. The second machine's replication approach is based on periodic replication of checkpoints, i.e., processor and memory states, with a high frequency [55]. This replication method is appropriate for SMP environments, but requires more bandwidth for the replication process.

Virtualization also reduces virtual machine performances. A detailed analysis of XEN hypervisor's scheduler identifying bottlenecks for media applications is presented by M. Lee et al. [56]. Modifications aiming at a better adaptation of the XEN scheduler for media applications are presented by M. Lee et al. [57]. These two works show that virtualization supports performance requirements of VoIP servers. Adaptation of virtualized environment for high availability VoIP servers is proposed by D. Patnaik et al. [58], where authors show that the real-time replication without network buffering performs well enough to run a VoIP server. Nevertheless, data consistency isn't preserved without network buffering; primary and backup machine states can become desynchronized. The main virtualization's advantage is its transparency at the application level, i.e., any VoIP server implementation can be used.

In order to support complex VoIP services, servers are composed of many modules. Therefore, the application level replication approach becomes too complex and security requirements are hard to meet [43]. On the other hand, the migration of virtualized machines is transparent at the application level and preserves TCP/TLS connections.

## **4.2 Problem Analysis**

Virtualization is a new challenging mechanism and its use for real-time and multimedia applications is being actively investigated by researchers and industry. In this study, I focus on network metrics such as jitter and burst generation, as these aspects are strengthened by the replication mechanism itself.

### **4.2.1 Virtualization Basics**

Virtualization allows one or more virtual machines to run simultaneously on one hardware platform. Hardware sharing is ensured by a specific layer between the hardware and virtualized machines called the hypervisor. Each machine disposes of several resources that are attributed by the hypervisor and the machine can be unaware of being virtualized. Hypervisors allow migration of virtual machines between different hardware servers in a shutdown state. Current hypervisor implementations improved the migration mechanisms and enable a real-time migration without virtual machine shutdown. This approach is called live migration. The live migration is useful to change the server hosting the VM to

obtain more resources or to enable hardware maintenance. Another usage is to build high availability systems. The live migration process can be described as follows. The virtual machine execution is paused by the hypervisor; the virtual machine's state (including memory content and processor state) is transferred to the backup server where the virtual machine is resumed.

#### 4.2.2 Continuous Live Migration

To overcome hardware faults, live migration has been improved to support continuous live migration. This type of migration allows immediate resuming of a backup virtual machine when the primary server goes down or becomes unreachable. The execution of the primary virtual machine is divided into small periods of time and the state of the primary machine is entirely replicated on the secondary server at the end of each period. These periods are called epochs. The backup machine on the secondary server waits in a paused state. The replication period depends on the machine purpose and typically ranges from 40ms to 200ms [55]. In case of a failure, the backup machine on the secondary server is resumed at the last completed checkpoint. This means that a packet loss will be observed from the user's perspective.

##### 4.2.2.1 Data Consistency

The challenge of live migration mechanism is to ensure data consistency in case of primary server failure. To minimize the interruption and to preserve performances, the replication is being done continuously.

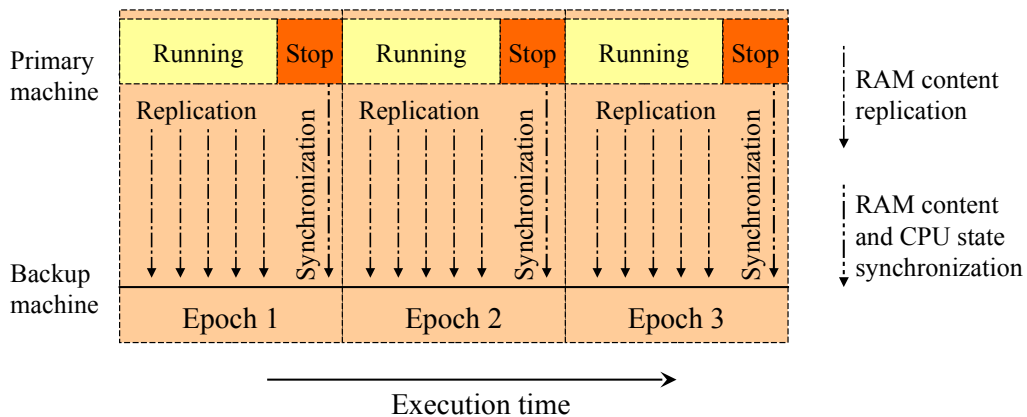


Figure 6: Network buffering during continuous live migration

A synchronization checkpoint ensuring the consistency is introduced at the end of each replication cycle. The synchronization checkpoint consists of stopping the execution, synchronization and consistency verification and resuming the execution [55].

To integrate such a system into the real environment, network outputs need to be saved in a buffer between each checkpoint and released once the state of virtual machine is successfully replicated on the backup server. This buffering process introduces jitter and packets burst. This mechanism is depicted in Fig. 6. Although these events can be tolerated in interactive client/server applications, it is a significant issue for real-time applications such as VoIP services. In certain cases, the buffering impact can be accepted for signalization, but needs to be eliminated for user data like RTP flows.

### **4.2.3 Proposed Solution**

This section details the proposed solution to the problem described above, including a possible way of its implementation.

#### 4.2.3.1 Principle

Current replication implementation buffers all outgoing packets (signaling, data). The buffering of real-time transport protocol packets has a negative impact on real-time applications. Thus, the replication mechanism needs to be enhanced by inspecting packets and classifying them in two categories: i) packets that need to be buffered and ii) packets that can be sent without being buffered. As the focus is on the VoIP service, this classification can be simplified as follows. All TCP packets are buffered in order to ensure TCP state consistency and to avoid signaling packet loss when using TCP or TLS protocol. Non-TCP packets are sent to the network without any processing by the hypervisor. This approach minimizes a negative impact on network forwarding such as latency increase, jitter introduction or burst generation. This classification needs to be refined to match only packets with real-time data.

#### 4.2.3.2 Implementation

In the testbed, the latest XEN hypervisor version 4.2-unstable [59] is used. A server with XEN hypervisor includes the following components:

- XEN hypervisor.
- Domain 0 – the XEN's terminology for privileged domain with direct hardware access run by the hypervisor to manage the server and control other virtual machines.

- Domain U – the XEN’s terminology for unprivileged domains hosting virtualized machines.

The XEN hypervisor live migration implementation is provided by Remus project [55]. Remus provides fault detection, virtual machine state (memory and processor state) replication and network buffering. The network buffer implementation is based on Linux traffic control together with Intermediate Functional Block [60]. This component allows ingress traffic queuing. The traffic sent by virtual machines is buffered at the Domain 0. Standard Linux traffic control tools are used to redirect all incoming traffic from the replicated machine to the buffer. Each transferred packet is analyzed and only TCP packets are redirected to the buffer. The traffic classification can be done directly by the Linux kernel. Thus, the kernel code doesn’t need to be modified. The implementation proposal consists of the following 3 steps:

- Packet classification using Linux packet inspection.
- Introduction of redirection rules to buffer only non-real time packets.
- Real-time packets are transmitted without any buffering.

#### **4.2.4 Obtained Results**

This section presents the results of the measurements performed.

##### 4.2.4.1 Testbed Description

A testbed platform with low performance machines and 100 Mbit/s network switch is implemented. One server includes a double core Intel Pentium-M processor running at 1.8 GHz (3 GB RAM), whereas the second server is built on a single core AMD Duron processor running at 900 MHz (1 GB RAM). Both servers are running XEN hypervisor, version 4.2-unstable and Linux Ubuntu 12.04 [61] with 3.2.0-24-pae kernel in Domain 0 and the same kernel in Domain U. Fig. 7 illustrates the testbed network configuration. Additionally, Freeswitch [62] version 1.0.7 as a SIP (Session Initiation Protocol) server is used together with one SIP VoIP phone and one SIP softphone. The voice signal processing is done using G.711 A-law codec with 20 ms packetization time (ptime).

##### 4.2.4.2 Jitter Calculation

As there is no packet loss except during the effective migration, the jitter values are considered to evaluate the impact of virtualization. Jitter calculation is implemented as described in [14]. It is a first-order estimator with noise reduction using a gain parameter.

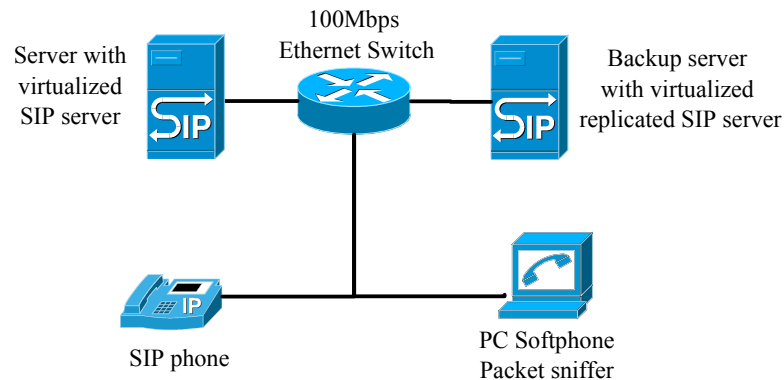


Figure 7: Testbed configuration

#### 4.2.4.3 Virtualization Impact

Figs. 8 to 12 show the jitter observed under different conditions. The horizontal axes represent time and vertical axes jitter. As shown in Fig. 8, when running a SIP server without virtualization, the jitter is about  $100\mu\text{s}$  with a minimal variation. Performances of Freeswitch running on a virtualized machine (Fig. 9) are not as good as without the virtualization, but still lower than  $500\mu\text{s}$ , which is usually the limit in the Service Level Agreement among operators.

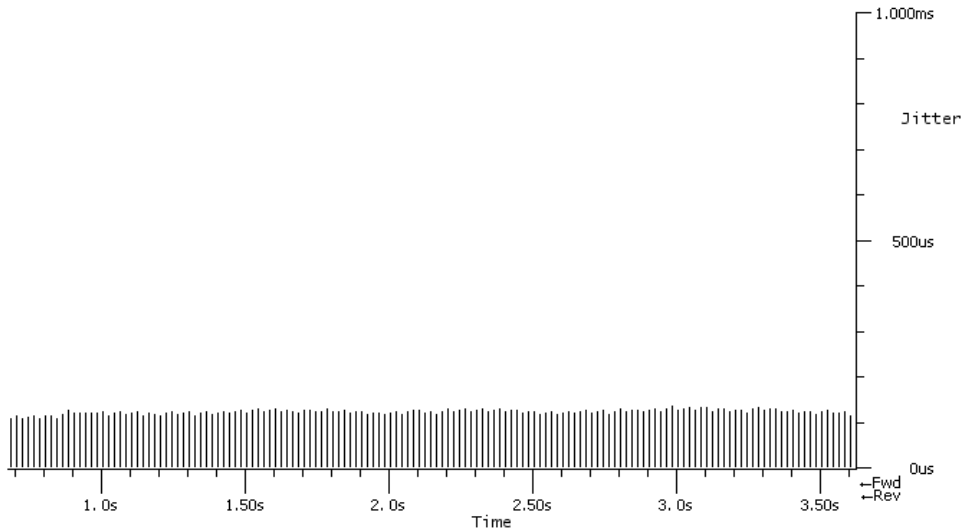
#### 4.2.4.4 Live Replication Impact

Fig. 10 illustrates the impact of continuous live migration process on the machine performances. Without any modifications, network buffering introduces an important jitter and packet bursts at each checkpoint. A peak can be observed approximately every 400 ms, which equals to the checkpoint interval.

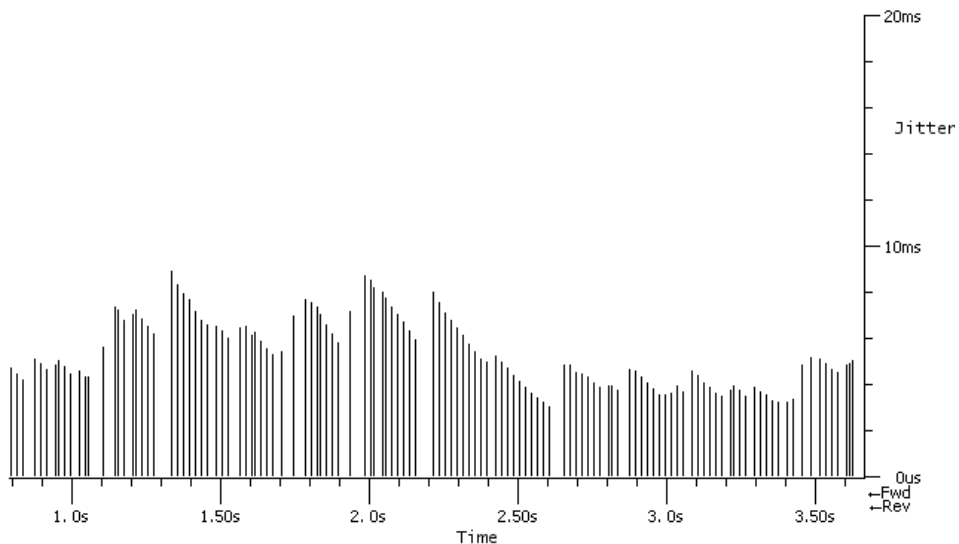
These peaks correspond to a burst of data released from the buffer. With a 20 ms packetization time, each peak represents about 20 RTP packets by call and direction. The observed degradation of network characteristics deteriorates quality of VoIP calls. Such packet bursts can overload network equipments and cause packet loss. The impact of the continuous live replication is studied in more detail in [63].

The proposal's impact is shown in Fig. 11. Contrary to the unmodified replication mechanism, RTP packets are forwarded as long as the virtual machine runs. The observed jitter is more important with continuous live replication because of performance impact generated by the continuous VM state replication. This effect is emphasized by the low-performance CPU. Execution interruptions can be observed every 300ms, which

corresponds to the configured checkpointing interval. The considered testbed is composed of relatively low performance machines and a single 100 Mbit/s network, while a 1 Gbit/s network dedicated to replication is recommended. Used low performance testbed allows us to verify the behavior of the proposed modification with limited resources.



*Figure 8: Jitter without virtualization*



*Figure 9: Jitter measured calling via a virtualized Freeswitch server*

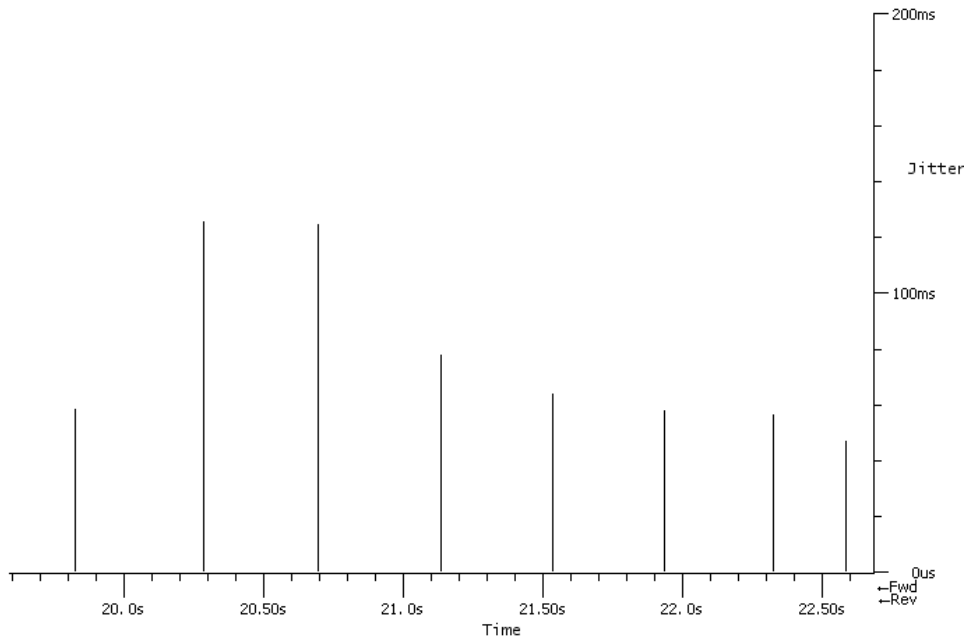


Figure 10: Jitter using unmodified replication mechanism

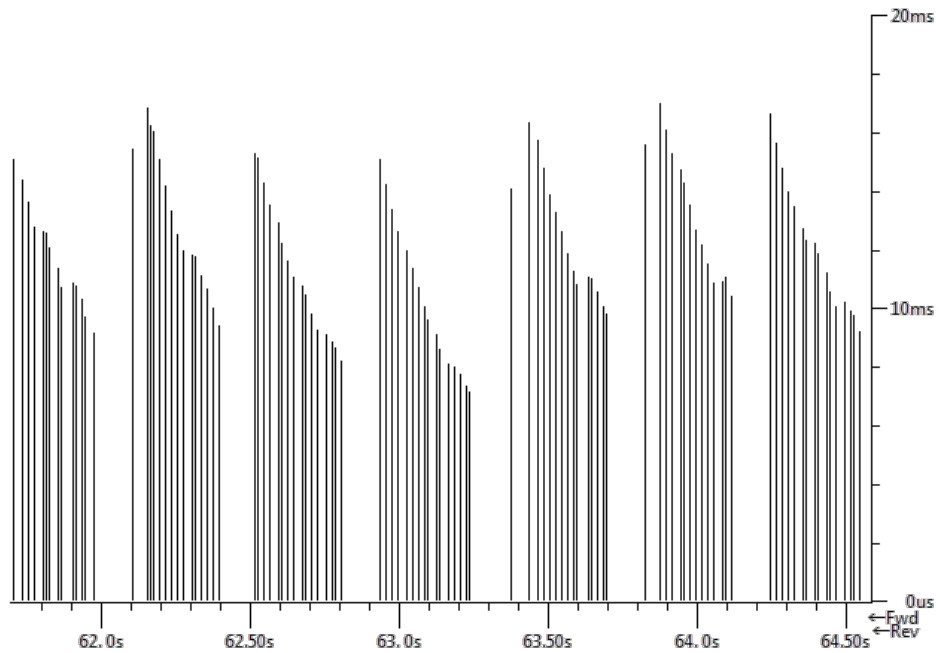


Figure 11: Jitter using modified replication mechanism

Measured jitter and interruption length are therefore quite high, but fully rectified by jitter buffer and the impact on the call quality is unnoticeable from the user's point of view.

Note that the calculation method defined in [14] considers the previous packet jitter; therefore, a descending trend following each interruption can be observed.

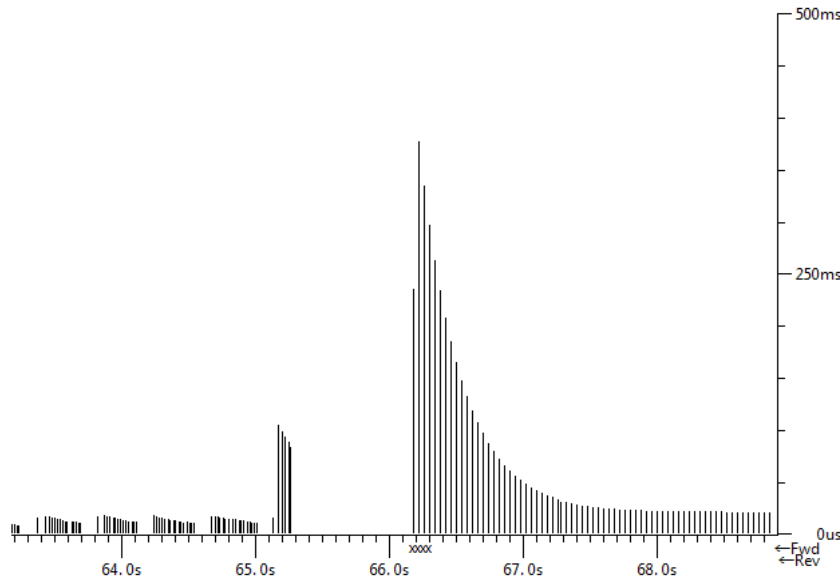


Figure 12: Jitter and packet loss during the migration following a failure

The objective of the continuous live migration is to maintain established calls and connections in case of hardware or network failure. Once a fault is detected, the replicated machine resumes on the backup server without call interruption. To announce the new location of the virtual IP address, a gratuitous Address Resolution Protocol request is used.

The jitter observed during the migration is depicted in Fig. 12. The system in a stable state is presented on the left side, continuous VM replication running. The failure of the primary server is represented by a period without packets, which lasts about 1s. The detection of the primary server's failure, generation of the gratuitous ARP and its processing by network components is time consuming. In the presented configuration it's almost 1s, but the time depends strongly on the network hardware. Users of the VoIP service perceive the failover as a short interruption. As the second virtual machine runs without continuous live replication, the jitter observed after the migration is stable without interruption. The descending trend is due to the used calculation method discussed above.

### 4.3 Conclusion and Future Work

Recent VoIP systems are complex and often interconnected with external services. The possibility of implementing a high availability system with no impact on the application is therefore a very challenging task. In this section, the jitter is studied as a major drawback



of virtual machine's live migration. The modification proposed improves networking properties of this mechanism without impacting the data consistency. The proposition is easy to implement and its impact on the system performance is negligible compared to the unmodified system. Presented measurements demonstrate that the modification is beneficial for VoIP and other soft real-time applications. Contrary to the conventional implementation, the implementation does not introduce any jitter to the real-time packet flow except the jitter caused by the interruption required to synchronize replicated virtual machines.

The remaining jitter is caused by the interruption necessary for checkpointing. The optimal length of checkpointing interval is another point that is worth to be investigated as well as better packet classification.

## 5 Actors

### 5.1 Introduction

Software fails because of errors called bugs. According to [64], there are two types of bugs: Bohrbugs and Mandelbugs. Bohrbug is a bug that occur systematically with a given set of entry data. Mandelbug "is a bug whose underlying causes are so complex and obscure as to make its behavior appear chaotic and even non-deterministic" [65]. Examples of mandelbugs are race conditions or software aging-related bugs. Contrary to bohrbugs, which are relatively easy to reproduce, the cause of mandelbugs can be hard to figure out and therefore mandelbugs can be hard to fix. Software can also fail because of a hardware failure [66].

In order to improve the availability of services proposed by the software components, different levels of fault-tolerance can be implemented. One of such techniques is called software rejuvenation [67]. It consists in restarting of software components to avoid software aging-related bugs. The main problem of software rejuvenation is when to trigger the restart. Authors in [68] show that main scheduling approaches used nowadays are time-based and threshold-based. The Fig. 13 shows resolved and unresolved bugs in the popular open-source IPBX Asterisk [32].



Figure 13: Resolved (green) and unresolved (red) bugs in Asterisk IPBX

In this section, a software architecture based on the actor model is studied. The actor model was introduced in 1973 [69] and has since been adopted by many programming languages, i. e. Erlang [70]. Recently, the model was implemented by the Akka toolkit, which is a framework focused on concurrent distributed applications [71].

Akka allows actor integration to a Java [72] or Scala [73] based software. An actor is a component providing some simple specific service. An actor based system is composed of an actor hierarchy. Actors are organized in a tree structure and communicate using immutable messages. The top of the tree is the main actor and each actor supervises its children. Therefore, actor base systems are fault tolerant, scale well and can be distributed easily. Furthermore, actor systems are event-driven, which fits well telecommunication systems. An implementation of a VoIP server implemented in Scala with Akka is proposed. However, the associated availability model is also presented. The software architecture was elaborated with a focus on the availability and resilience. Actor model allowed us to implement a partial rejuvenation mechanism and thus to improve the overall server availability.

## 5.2 Related Work

VoIP system rejuvenation is studied in [74]. The proposed availability model is based on semi-Markov process. An optimal rejuvenation time maximizing the system's availability is found using stochastic analysis of the proposed model. The work is focused on the rejuvenation of the whole system, partial rejuvenation is not considered.

The availability model presented in [75] addresses multi-level fault recovery. Component-based system is modeled using Discrete Time Markov Chain (DMTC) and the model is then extended to reflect different recovery possibilities, namely restarts, retries, reboots and repairs. The model is applied to an example and numerical results are given. The resulting model can be used for designing systems with fault tolerance implementation at different levels. Component-based software systems are a good analogy to the actor systems. Nevertheless, an actor can be restarted without the need to restart the whole component. Therefore, it can't be modeled by the technique proposed.

Another multi-level fault recovery availability model is presented in [76]. This model uses semi-Markov processes. Considered corrective actions are minimal repair,

major repair and restart. The system state in the proposed model is global; therefore, it's hard to apply on component or actor based systems.

The work presented in [77] introduces an extension to the Rebeca language [78] called pRebeca. Rebeca is a high-level actor based language for distributed system's behavior modeling. pRebeca extends Rebeca with following probabilistic features: alternative behaviors and message loss. Rebeca and pRebeca aim to facilitate behavior modeling and checking, but does not address the availability computation.

Authors in [79] gives an overview of Erlang's features for building fault-tolerant applications. Reliability qualities of Erlang language like fast process creation and tear-down, interprocess linkage, distribution and live upgrade are detailed. The article include neither concrete examples nor availability model.

A native actor implementation for C++ is presented in [80]. A detailed implementation's description is presented together with performance evaluation. This work does not include any availability modeling.

### **5.3 Introduction to Akka**

The Akka toolkit is a software library for concurrent and distributed applications [71]. It's an open-source project available under the Apache 2 License [81]. The toolkit is still in active development. Two programming languages are supported: Java and Scala. Akka is using the actor model to provide parallelism, event-driven programming model, scalability and fault-tolerance. Actors are very lightweight, according to Akka's documentation [71], more than 2500 actors fits in 1MB (megabyte) of RAM (Random Access Memory). Moreover, actor's creation and termination are fast and inexpensive.

Lightweight actors allow Akka to adopt the "let it crash" strategy [70]. An actor unable to successfully terminate requested task should crash and the error is to be resolved using its supervising actor. It encourages the separation of concerns principle introduced by E. W. Dijkstra in 1974 [82]. This principle advises focusing on each aspect independently from the others in order to work effectively. In the software developed with Akka, the code dealing with error is clearly separated from the business logic. Furthermore, actor model based applications fit well with multi-core computing [83].

Akka also provides modules for remoting, scheduling, input/output processing, logging etc. Akka based applications run as Java or Scala applications and therefore they can take advantage of the existing Java and Scala ecosystem like libraries and development tools. Also, standard profiling and debugging tools can be used. Furthermore, Typesafe console allows developers to monitor an Akka based system. The Akka toolkit and Console are supported by Typesafe [84].

### 5.3.1 Actor Model

According to Akka's documentation: "Actors are objects which encapsulate state and behavior, they communicate exclusively by exchanging messages which are placed into the recipient's mailbox." [71]. Actors are instantiated in a hierarchical structure in the form of trees. Each actor supervises its children and can also supervise other actors. This type of supervision allows avoiding defensive programming; a failure can be processed by a supervising actor or escalated to the supervisor's supervisor. An example of such hierarchy is depicted in Fig. 14.

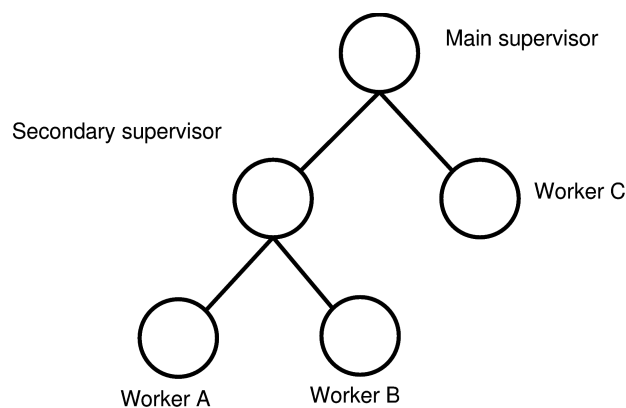
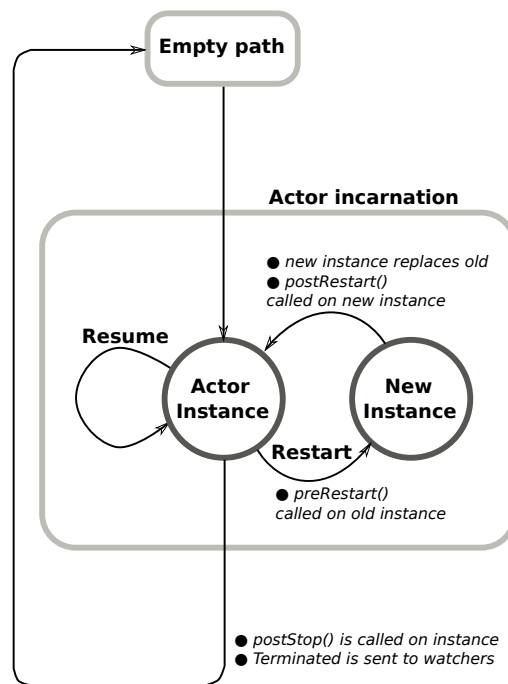


Figure 14: Actor system tree

### 5.3.2 Actor's Life-cycle

The Akka toolkit provides a set of functions to execute some customized operations when a new actor is created, stopped or during a restart of an existing actor. These functions can be used to restore actor's state from external data storage or release acquired resources. The actor's life cycle with associated functions are depicted in Fig. 15.



[adapted from <https://github.com/akka/akka/tree/master/akka-docs>]

Figure 15: Actor life-cycle

### 5.3.3 Supervision, Monitoring and Fault Tolerance

Each actor is notified when one of its child actors is terminated, whether it's a normal termination or due to a crash. An actor's behavior on this notification is called supervision strategy and is fully customizable. Customized supervision strategies are useful mainly for two reasons. It allows implementation of fault-tolerant patterns like error kernel [85]. Also, it's easy to create a parallel supervising actor tree. Besides its child actors, each actor can supervise any other actor in the system as needed.

### 5.3.4 Finite State Machines

Akka integrates a support for the Finite State Machine (FSM) based on the Akka actors. Telecommunication system's software is processing user data and control messages. By definition, it's event based software. Therefore, state machines support facilitates development of telecommunication software. Each machine is implemented as an actor. Hence, it can be supervised as other actors. Furthermore, it's pre-start methods can be used to restore the internal state when an actor is restarted following a crash.

## 5.4 SIP Server Implementation

A SIP server implementation is presented in this section. The implementation uses Akka actor framework and is developed using the Scala language. Since the SIP parser is not studied in this work, a third party open source SIP parser is integrated. The proposed server implements registrar, location service and a stateful proxy.

### 5.4.1 Component Model

In order to give an overview of the software architecture, a component model describing how the server software is structured is presented. Encoding, transport and transaction layers are common to the components developed. These layers are linked together in one unique module called Transport. There are two transaction user layer modules: registrar and proxy. The location manager is basically just a database with an appropriate interface; therefore, it is implemented as a standalone component. This architecture is depicted in Fig. 16.

### 5.4.2 Actor Model

In this section, the architecture of the actor system and interactions between actors are explained.

#### 5.4.2.1 Transport Module

Due to the complexity of the TCP and SCTP protocols, the prototype is limited to the UDP transport. An open source SIP parser is integrated. As this research work is focused on the availability brought using actors, these choices are not limiting.

The transport module provide UDP input/output functions, SIP message encoding and parsing and it also deals with transactions. The UDP input/output is implemented using two actors. The first is acting as a socket listener and the second as a sender. These actors interact with another actor providing parsing and encoding of the SIP protocol messages. Decoded messages are passed to the actor called Transaction router. The router maintains a table of existing transactions and it routes the message to the corresponding transaction. Each transaction is represented by an actor based finite state machine. If the transaction doesn't exist yet, it is created. Also, when a transaction terminates, it is removed from the table with existing transactions.

#### 5.4.2.2 Registrar and Location Module

The registrar module is implemented as a simple actor translating the messages from the register transactions to the location manager. The location manager as a simple actor with

an internal data storage, which emulates a database or another data storage possibly used in production ready applications.

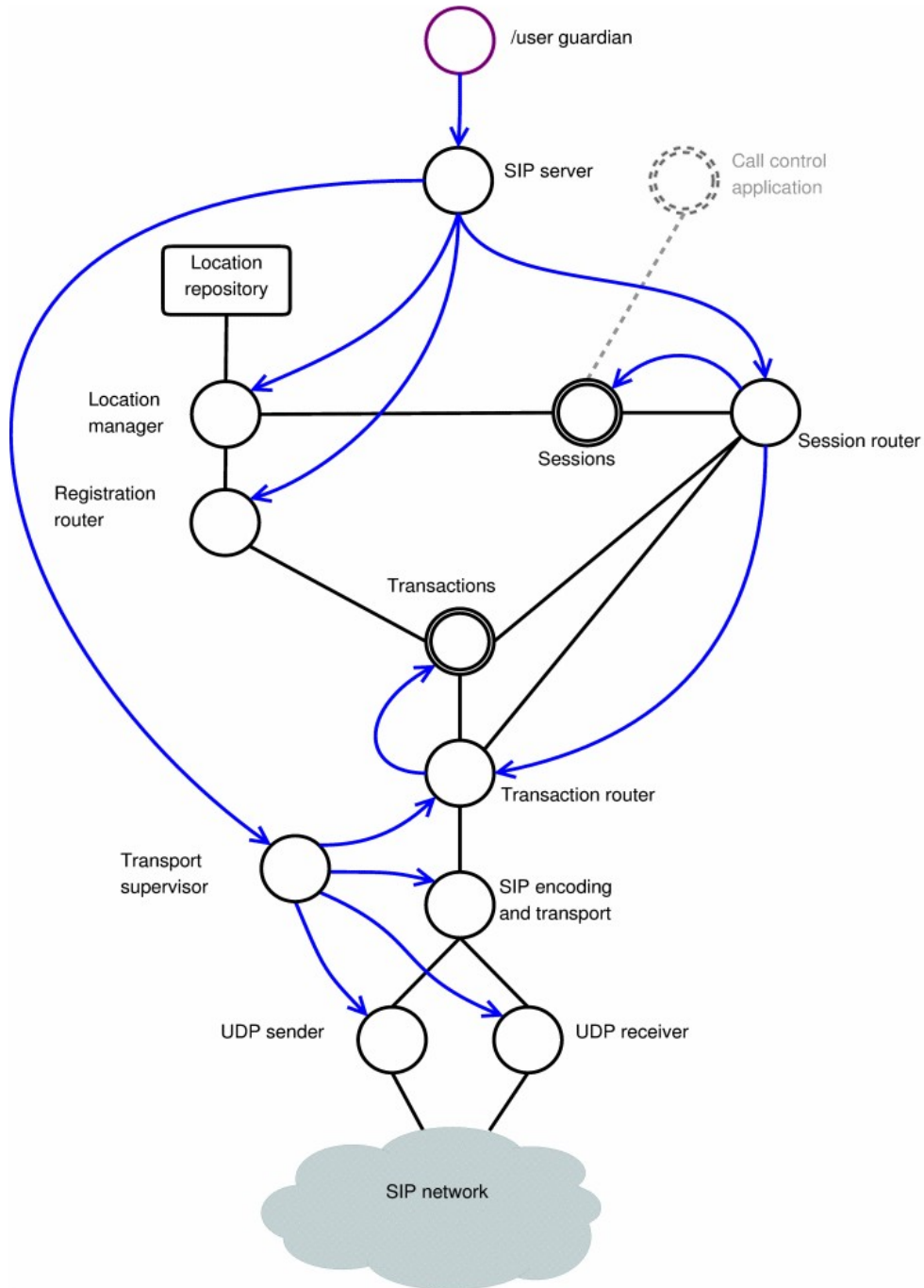


Figure 16: SIP server component model



#### 5.4.2.3 Proxy Module

The main actor of the proxy module is a session router. A session is defined as an association of two dialogs, simply as a call. The session router handles dialogs and sessions. A new session actor is spawned for each dialog. Subsequent dialog messages from the transaction are forwarded to the corresponding session actor thanks to the table of correspondences maintained by the session router. The new session actor can ask the session router to create a new dialog. The session router then asks the transaction router to create a new transaction associated with the asking session actor. It also forwards to the associated actor dialog terminations. Each session actor deals with the dialog termination by terminating the corresponding dialog and thus the call, then it stops itself. On the session actor termination, the session router removes the actor from its table of correspondences.

### **5.4.3 Fault Handling**

The presented implementation focuses on fault tolerance. The architecture described above is designed with the actor fault tolerance principles in mind. The basic principle is to delegate the risky tasks to the lowest actors in the hierarchy. In the implementation, there is one actor called SIP server, which is the topmost user defined actor of the application. It is supervised by the user guardian. The SIP server creates and supervises Registrar router, Location manager, Session router and Transport supervisor.

#### 5.4.3.1 Registrar Router and Location Manager

The Registrar router is stateless and therefore can be simply resumed on a failure. The location manager is ideally only an interface to replicated data storage. The actor is thus simply restarted on the failure - the connection to the data storage should be restarted following a failure.

#### 5.4.3.2 Session Router

The Session router actor maintains a table of correspondences between dialogs and session actors. When an actor is restarted, the default behavior is to stop its children. The standard supervision strategy is used for the Session router. All its children are, therefore, stopped when the router is restarted. Also, the Session router watches the Transaction router in order to correctly process its possible restart.

#### 5.4.3.3 Transport Supervisor

This actor's role is to supervise a set of actors implementing the transport module functions. Actors UDP sender, UDP receiver are stateless and can be restarted on failure

without losing any saved state. Restart is preferred to resume as it's important to reopen the socket in case of reconfiguration such as an IP address update. SIP encoding and transport is stateless and resumed on failure. It is considered that the parser error doesn't require actor restart. The transaction router maintains the directory of transactions and associated actors. Transaction actors are its children. These transactions are stopped when the transaction router is restarted.

#### **5.4.4 Timing Constraints**

It was specified whether each actor is restarted or resumed, but if the problem persists, the action would be executed over and over again. To avoid this infinite loop, the supervision strategy is extended with timing constraints. For the sake of simplicity, the same configuration is applied to all actors. If there are more than 5 failures in 30 seconds, the actor is stopped and the problem is escalated to the actor's father.

### **5.5 Availability Model**

The objective is to find out the advantage of using actors compared to a monolithic application. The behavior of each actor needs to be modeled - mainly the ability to restart or resume. Also, the timing constraints, i. e. the limited number of restarts within a given time period, must be reproduced by the model. Finally, the model must be scalable to fit complicated architectures.

Each actor includes following functions: a message processing part, the ability to create and supervise other actors. The actor's model has to expose corresponding interfaces. It should be noticed that actors work with two types of data: user data-and fault tolerance model data. Therefore, the model should be able to separate events of each type. These requirements are best fulfilled by stochastic colored Petri nets (SCPN). The stochastic colored Petri nets allow us to create a configurable modular model, where actors are represented by reusable modules. Colored token are used to distinct between user data tokens and restart control tokens.

#### **5.5.1 Single Actor Model**

First, a model of a single actor is to be created. This model can then be used to construct a model of the considered system. The presented proposition models an actor with following supervision strategy: the actor is restarted on each failure except if there are more than 3 failures in 30 seconds. In this case, the actor is stopped and the failure is propagated. The Petri net modeling actor behavior is depicted in Fig. 17.

There are six places serving as connection points when the model is used as a submodel. The first two places Start and Stop are life controlling places. Then there are two places for child actor lifecycle control named Restart and Poison Pill and finally two places for fail notifications, Failed for a notification from the child actor and Fail for upwards failure notification. An actor must be started by a token in the Start place. Then the transition T5 fires and a token is passed to the Idle place and Restart place. The Restart place is to be used to start child actors. The transition T1 fires when the actor fails. The token from the Idle place passes to the Fail place, the actor is stopped and no more jobs can be processed until the actor is restarted. The place called Failed receives tokens when a child actor fails. There is an auxiliary transition T6 allowing restart count processing. When a child fails, there are two possibilities. Either the transition T2 is fired and the child actor is restarted or, in the case that there were 3 restarts during the last 30 seconds, the child actor stays stopped and the failure is propagated upwards by a token in the Fail place (the transaction T3 is fired). Transition T2 is enabled only when the actor is started and there are less than 3 tokens in the RestartCount place. The transition T7 ensures the reset of the RestartCount. It fires each 30 seconds and takes one token from the RestartCount. This mechanism can be easily adapted to reflect the resume or any other strategy. If the actor doesn't have any children, this part of the model can be omitted.

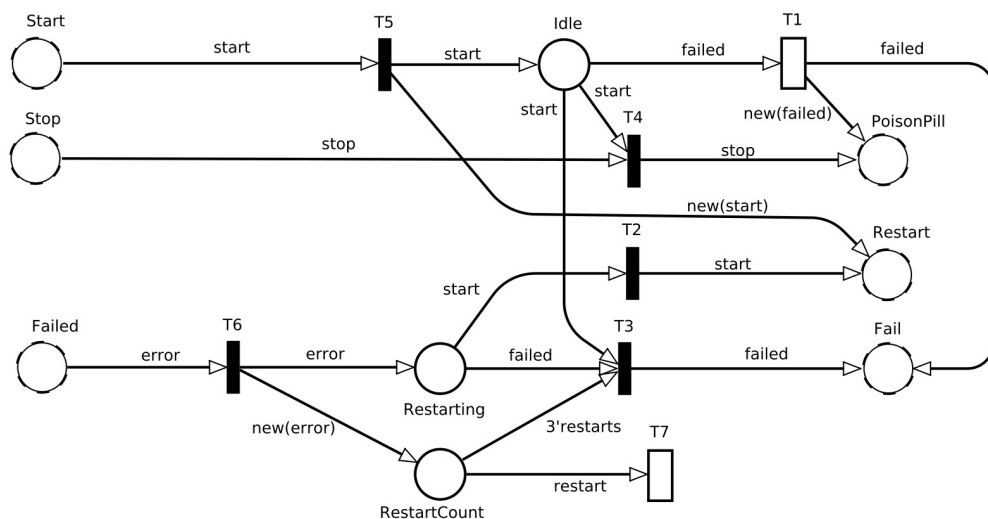


Figure 17: Single actor model

The proposed model can be used as a module for construction of high-level models.

### 5.5.2 SIP Transport Model

In this section, a model of the SIP server's transport layer is described. It is based on the single actor model presented above, which is used as a module. The Fig. 18 presents the module interface.

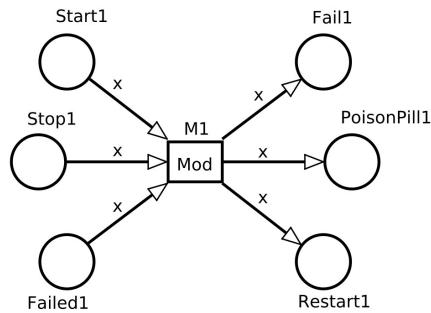


Figure 18: Single actor module interface

The module is configurable; hence, it is possible to reuse the same model with a customized parameter of failure distribution.

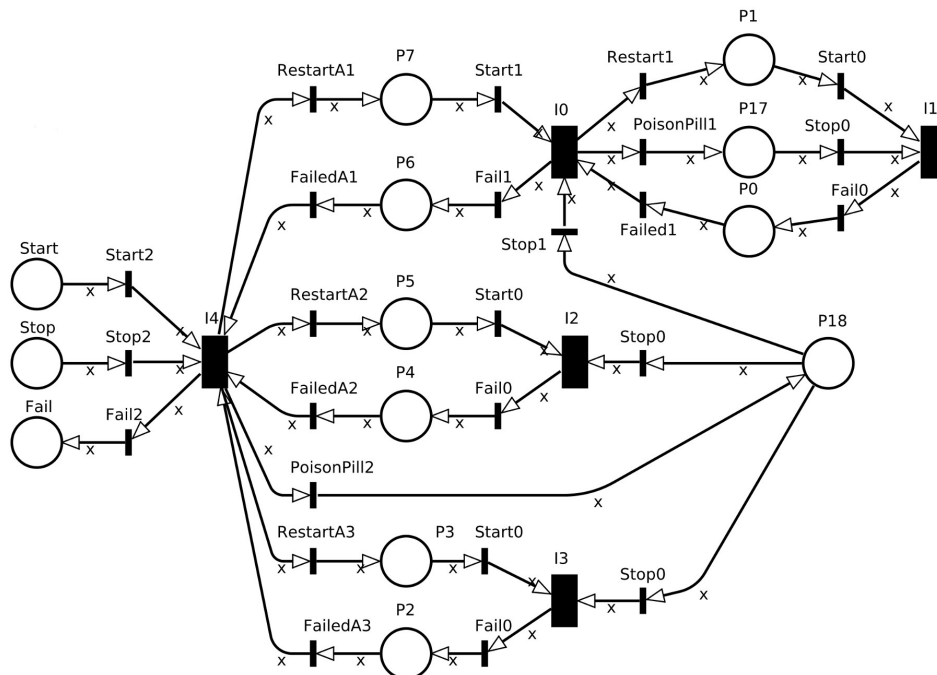


Figure 19: Transport layer availability model

The model of the transport layer composed off the presented module is shown in Fig 19. Each module instance is represented by a filled rectangle and name I0-I4. The module was adapted to expose the right number of child supervision interfaces.

## 5.6 Availability Analysis

This section describes how to calculate a system's availability using the model presented in the previous section.

A method similar to the one used in [86] is used. It consists in state qualification in a failure state set and an operational state set. Each state in the model is marked as failure or operational and the mean time spent in failure and operational states set is used to determine failure and repair rates. In the presented case, failure states are Restarting and Restart states of each actor. The probability of the failure state can be written as:

$$P_{Failure} = \sum_{i=1}^n P_{i,Restarting} + P_{i,Restart} \quad (1)$$

Where  $i$  determines the number of actors in the system and goes from 1 to the total number of actors in the system,  $P_{i,Restarting}$  and  $P_{i,Restart}$  are probabilities of each failure state by actor instance. Likewise, operational states are  $P_{i,1}$ ,  $P_{i,2}$  and  $P_{Start}$  and the probability the operational state can be expressed as:

$$P_{Operational} = \sum_{i=1}^n P_{i,1} + P_{i,2} + P_{i,Start} \quad (2)$$

A basic model for the system's availability is a Markov chain with two states: UP and DOWN. The system gets from the state UP to the state DOWN with a failure probability  $\lambda$  and back to the UP state with a repair probability  $\mu$ . This model is depicted in Fig. 20.

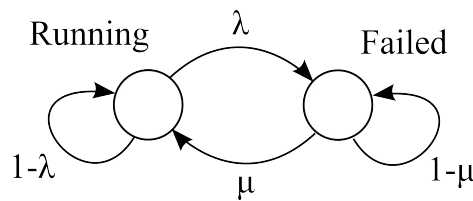


Figure 20: Two-state Markov chain

Let's define the steady-state availability as follows:

$$A = \frac{Uptime}{Uptime + Downtime} \quad (3)$$

Using the failure and repair rates the equation (3) can be rewritten:

$$A = \frac{\mu}{\lambda + \mu} \quad (4)$$

The considered model is limited to two states, therefore, it can be stated that:

$$P_{Operational} + P_{Failure} = 1 \quad (5)$$

The reciprocal of  $P_{Failure}$  can thus be used as the failure rate  $\lambda$  and the reciprocal of  $P_{Operational}$  as the repair rate  $\mu$ . The steady-state availability can be rewritten:

$$A = \frac{P_{Operational}}{P_{Operational} + P_{Failure}} = P_{Operational} \quad (6)$$

The next section presents numerical results.

## 5.7 Numerical Results

According to results presented in [87] and [88], the time to failure of software application is lognormally distributed. Each actor can be seen as an independent application interacting with other application. Thus, it can be assumed that the actor's time to failure is also lognormally distributed.

Due to the utilization of multiple general transitions, a simulation must be performed to get numerical results. Simulations were performed by the TimeNet software tool developed at the Technische Universität Berlin [89].

The lognormal distribution is defined using two parameters: mean and standard deviation, respectively  $\mu$  and  $\sigma$ . The probability density function is defined as follows:

$$f_x(x; \mu, \sigma) = \frac{1}{x \sigma \sqrt{2\pi}} e^{-\frac{(\ln(x) - \mu)^2}{2\sigma^2}}, x > 0 \quad (7)$$

If expected value and variance are known, parameters can be calculated using the following equations:

$$\sigma = \sqrt{\ln\left(1 + \frac{\text{Var}[X]}{(E[X])^2}\right)} \quad (8)$$

$$\mu = \ln(E[X]) - \frac{\sigma^2}{2} \quad (9)$$

Based on characteristics presented in [90], it is estimated that an actor fails each 92 days (an arbitrary expected value corresponding to 2208 hours). According to [90], an appropriate value standard deviation for telecommunication systems is  $\sigma=0.4$ . TimeNET time unit is second, the mean is calculated using the equation (9):

$$\mu = \ln(2280 \cdot 60 \cdot 60) - \frac{0.4^2}{2} = 15.81 \quad (10)$$

The hazard rate of proposed lognormal distribution is shown in Fig. 21

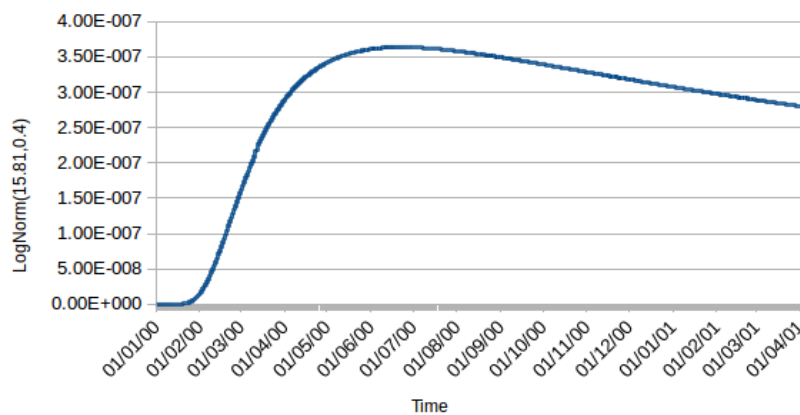


Figure 21: Hazard rate of the lognormal distribution  
with parameters  $\mu=15.81$  and  $\sigma=0.4$

To compare the availability of the proposed actor based implementation with some usual implementation, a reference model reflecting the availability of comparable software without actors is proposed. The model is depicted in Fig. 20.

Each actor is transformed in a corresponding section represented by a transition with the same availability as the actor. A failure in one of those sections makes the system fail and restart (the token moves from the Running place to the Failed place). Such a failure can be observed when a java application fails because of an unanticipated exception while in an actor system only the actor fails and is restarted. Lognormal distribution with previously defined parameters models both an actor's failures (transition T1 in Fig. 17) and a module's failures (T28-T33) in Fig. 22. A repair rate between 30 seconds and 5 minutes is assumed. TimeNET model time unit is seconds, the repair rate is thus defined by uniform distribution with parameters  $a=30$ ,  $b=300$ .

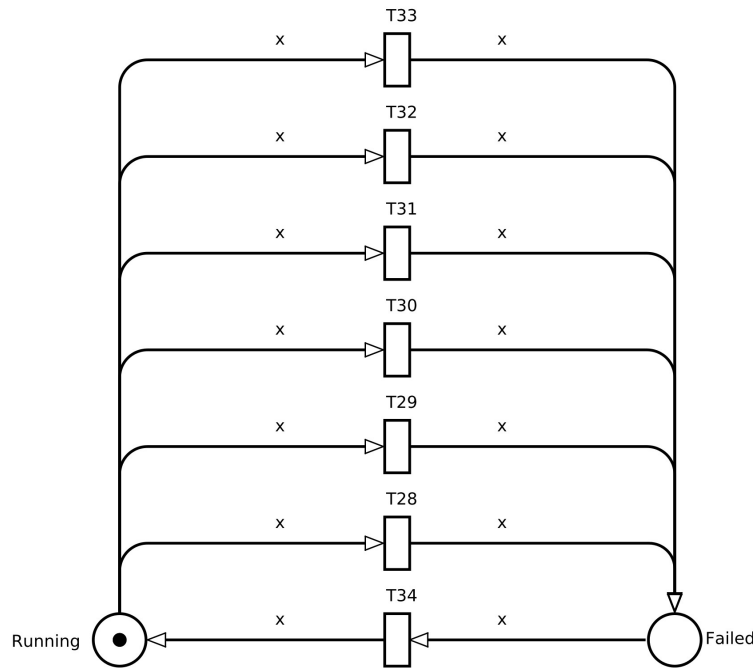


Figure 22: Availability model of considered standard implementation

A set of simulations is performed to evaluate the availability of both models. The table 1 summarizes the results. Actor based system outperforms the standard one by a magnitude of two orders. In terms of downtime the difference is more noticeable, 5 hours and 41 minutes for the actor based system opposed to 4 days and 15 hours for the standard one.

System	Availability	Downtime/year
Actor based implementation	0.9993507	5 hours 41 minutes
Standard implementation	0.9872581	4 days 15 hours

Table 1: Availability comparison between an actor based system and a standard one

Based on the proposed model, the impact of the failure rate  $E(X)$  on the system's availability is studied. The table 2 provides availability values for three different simulations. The expected value is divided by 2 from one simulation to another. For the actor system the impact is minimal and is disguised by the simulation precision. It's expected because the actor model handles failures and the system fails rarely. The impact is more noticeable on the standard system.



Failure rate	Actor based system	Standard system
E(X)=2208 hours LogNormal( $\mu=15.81$ $\sigma=0.4$ )	0.9993507	0.9872581
E(X)=1104 hours LogNormal( $\mu=15.11$ $\sigma=0.4$ )	0.9998218	0.9736801
E(X)=552 hours LogNormal( $\mu=14.42$ $\sigma=0.4$ )	0.9997122	0.9550325

Table 2: Impact of the failure rate on the availability

## 5.8 Conclusion

This part of the work describes the advantages of the actor model for SIP server software. A SIP proxy implementation is proposed and analyzed. The implementation is based on state-of-the-art technologies, the Akka framework and the Scala programming language. A modular availability model of the presented software is built using hierarchical stochastic colored Petri nets.

The proposed model is modular and each module is configurable, which allows modeling of complex software architectures. Furthermore, the model uses non exponentially firing times to reflect software failure distributions. A model of a non-actor software implementation is proposed for comparison with the usual approach. Using these models and TimeNet simulation software the overall availability is analyzed and compared. The actor based implementation outperforms the standard one by a magnitude of two orders. The fault-tolerance is confirmed by the fact that the actor model based implementation availability is less dependent on the failure rate than the standard implementation.

The proposed model with failure rates from a running system can be used to predict a system's availability.

## 6 Contributions of the Thesis

The main contributions of this thesis are:

- A detailed survey of existing solutions and architectures for highly available VoIP systems.
- An analysis of the continuous live migration mechanism enabling an immediate failover of a running virtual machine in case of a hardware or software failure. The work demonstrates that the actual implementation doesn't meet requirements of real-time systems like VoIP servers.
- An improvement in the continuous live migration is described, implemented and validated. It resolves the problem of the jitter identified in the current implementation.
- A study of the actor model in the context of high availability software architectures. A highly available implementation of a SIP proxy is proposed. High-availability is reached thanks to the mechanisms enabling fault-tolerance which are also described.
- A new availability model suitable for actor based systems is elaborated and used to estimate the availability of the proposed implementation. The model can be used for any actor based system.
- A reference model for a traditional implementation is designed. Using the reference model, the availability of the actor based implementation is compared with the availability of a traditional implementation.

## 7 Conclusions

This work analyses the robustness of VoIP systems, mainly in an IPBX environment. It focuses on the software architectures. The aim of this thesis is to find a software architecture allowing IPBX systems to reach the five nines availability standard for telephony systems. The work is divided into three parts: a survey of the highly available VoIP systems, a study of high-availability solutions based on the virtualization techniques and an analysis of the actor model in the context of the software development.

At first, solutions for the high-availability VoIP systems are analyzed. Requirements on the network architecture, network services and hardware components are detailed with references to standards and related articles. Standardized and emerging software solutions are examined. Advantages and drawbacks of each solution are underlined. The conclusion of these investigations is that there are industrialized solutions for the network and hardware infrastructure, but only partial solutions for the software. Therefore, the second and third parts focus on suitable software architectures.

The second part investigates the convenience of virtualization for high-availability VoIP systems. Besides flexibility, hypervisors also offer capacities for high-availability systems. The most important one is the continuous real-time replication, because it fits well hot standby configurations. This type of replication works as follows: a virtualized running server is backed-up by a continuous real-time replication on another physical machine. The backup virtual machine takes the place of the master in the case of failure. A testbed enabling continuous real-time replication of a virtualized VoIP server was realized and is presented. Measurements were performed to evaluate the impact of virtualization on VoIP systems. The results show that the current implementations of continuous real-time replication are not suitable for the real-time data. It generates an important jitter and therefore degrades the call quality. A modification of the replication mechanism is proposed and realized. Measurements performed with the improved version confirm that the modified hypervisor is suitable for real-time systems. A VoIP server can then be backed-up using the continuous real-time replication. As the whole machine state is replicated, all calls with associated contexts are preserved when a failure occurs. This solution is application transparent and is appropriate for VoIP systems where high availability was not considered in the initial design.

The third part deals with the actor model in software architectures. The actor model is an interesting alternative to the object-oriented development as it permits to separate the application's logic from the error processing and parallelization. Advantages of this approach are studied in two ways: a highly-available SIP server prototype is described and implemented and a modular availability model suitable for actor based systems is proposed. The prototype design is detailed and employed fault-tolerant techniques are described. The availability model is built on the stochastic colored Petri nets and is hierarchical. A generic module representing one actor is proposed. The whole system is then modeled as a hierarchical composition of modules. A model of the same system implemented using standard programming techniques is also provided. Both models are simulated and results compared. Results indicate that the actor model can improve software availability by a magnitude of two orders. The actor model is worth considering when a new VoIP application is designed.

Present work proposes two ways to implement high-available VoIP systems. As each one suits a particular context, these two propositions are complementary.

## 8 Future Work

Continuous live migration is transparent for the application and can, therefore, be easily applied to any existing system. Nevertheless, a few points need to be improved. A research work focusing on the efficiency of virtual machine state replication is still in progress. Replication requires a lot of bandwidth and processing power. The modification proposed in this work is applicable for other applications than VoIP systems. The implementation should be extended to support other real-time data flows.

The actor based software architecture is guided by the data flow and the supervision requirements. A further study of the proposed availability model needs to be performed to derive design principles improving the overall availability. Furthermore, emerging projects like Kamon [91] enable an in-depth monitoring of actor systems. It is possible to monitor actors' failures and improve the model precision with the collected data. Such a model can then be used to predict availability. More research is needed to confirm and describe possibilities of the prediction.

## 9 Publications

### 9.1 Publications Related to the Thesis

#### 9.1.1 Publications in Impact Journals

Hlaváček, J. - Bešťák, R.: Fault tolerant VoIP server based on the actor model. Submitted to *Annals of Telecommunications*.  
*authorship share: Hlaváček 50%, Bešťák 50%*

#### 9.1.2 Publications in Reviewed Journals

Hlaváček, J. - Bešťák, R.: Quality of Service Management in the Voice over IP Systems. *Access server* [online]. 2010, roč. 8., č. 01, s. 00003. Internet: <http://access.feld.cvut.cz/view.php?cisloclanku=2010010003>. ISSN 1214-9675. (in Czech).  
*authorship share: Hlaváček 50%, Bešťák 50%*

Hlaváček, J. - Bešťák, R.: Availability Model for Virtualized Platforms. *Advances in Electrical and Electronic Engineering*. 2013, vol. 11, no. 5, p. 316-320. ISSN 1336-1376.  
*authorship share: Hlaváček 50%, Bešťák 50%*

#### 9.1.3 Publications Excerpt in Web of Science

Hlaváček, J. - Bešťák, R.: Configuration of Live Migration for VoIP Applications. In *Proceedings of 15th Mechatronika 2012*. Praha: Czech Technical University in Prague, 2012, p. 187-190. ISBN 978-80-01-04987-7.  
*authorship share: Hlaváček 50%, Bešťák 50%*

#### 9.1.4 Other Publications

Hlaváček, J. - Bešťák, R.: Live Replication of Virtualized VoIP Servers. In *The Eighth International Multi-Conference on Computing in the Global Information Technology*. Silicon Valley: International Academy, Research and Industry Association (IARIA), 2013, p. 277-282. ISBN 978-1-61208-283-7.  
*authorship share: Hlaváček 50%, Bešťák 50%*

Michaux, J. - Buu, E. - Hlavacek, J. and Najm, E. An open-source platform for converged services. In *Proceedings of Principles, Systems and Applications on IP*

*Telecommunications* (IPTComm '13). 2013, ACM, New York, NY, USA, p. 1-8.  
ISBN: 978-1-4503-2672-8

*authorship share: Michaux 25%, Buu 25%, Hlavacek 25%, Najm 25%*

Hlaváček, J. - Bešťák, R.: Software Architectures for High Available Telecommunication Service Platforms. In *Knowledge in Telecommunication Technologies and Optics - KTTO 2010*. Ostrava: VŠB - TUO, FEI, Katedra elektroniky a telekomunikační techniky, 2010, p. 92-96. ISBN 978-80-248-2330-0.

*authorship share: Hlaváček 50%, Bešťák 50%*

Hlaváček, J. - Bešťák, R.: Improvements in the Availability of SIP Networks. In *Proceedings of the 2010 Networking and Electronic Commerce Research Conference*. Dallas, TX: American Telecommunications Systems Management Association Inc., 2010, p. 109-117. ISBN 978-0-9820958-3-6.

*authorship share: Hlaváček 50%, Bešťák 50%*

Hlaváček, J. - Bešťák, R.: Improvements in the Reliability of the Computer-Based Telecommunications Systems. In *Proceedings CD-ROM of Digital Technologies 2007*. Žilina: Slovenská elektrotechnická spoločnosť, 2007. ISBN 978-80-8070-786-6.

*authorship share: Hlaváček 50%, Bešťák 50%*

## **9.2 Unrelated Publications**

### **9.2.1 Other Publications**

Bešťák, R. - Hlaváček, J.: New Model for the Service Providers Called Capability Maturity Model Integration for Services. In *Digital Technologies 2009*. Žilina: TU v Žilině, 2009, vol. 1. ISBN 978-80-554-0150-8.

*authorship share: Hlaváček 50%, Bešťák 50%*

Hlaváček, J. - Bešťák, R.: Emulation of Wireless Links Using NetEm. In *Proceedings of VIIIth Conference KTTO 2008*. Ostrava: VŠB - Technical University of Ostrava, 2008, p. 38-42. ISBN 978-80-248-1719-4.

*authorship share: Hlaváček 50%, Bešťák 50%*

Hlaváček, J. - Bešťák, R.: Linux High Resolution Timers in NetEm Emulator. In *RTT 2008*. Bratislava: STU v Bratislave, 2008. ISBN 978-80-227-2939-0.

*authorship share: Hlaváček 50%, Bešťák 50%*



## 10 References

- [1] Zimmermann, H., "OSI Reference Model--the ISO Model of Architecture for Open Systems Interconnection." Communications, IEEE Transactions on. 1980, vol. 28, no. 4, pp. 425-432.
- [2] Orange Business homepage, <http://www.orange-business.com>, [retrieved: June 2014].
- [3] Intel Server Calculated MTBF Estimates, rev. 1, 2009, <http://download.intel.com/support/motherboards/server/sb/s3420gpmtdfcalculationrev10.pdf>, [retrieved: June 2014].
- [4] Kim, D. S., Machida, F., Trivedi, K.S., "Availability Modeling and Analysis of a Virtualized System," Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium on , vol., no., pp.365,371, 16-18 Nov. 2009.
- [5] Kanso, A., Lemieux, Y., "Achieving High Availability at the Application Level in the Cloud," 2013 IEEE Sixth International Conference on Cloud Computing, 2013, pp. 778-785.
- [6] Havemose, A., Ngan, C. Y. P., "Method and system for providing high availability to distributed computer applications," Google Patents, 2013. US Patent 8,433,951.
- [7] Beyersdorf, C. F., Wermser, D., Hartmann, D., Cao, X., "Virtualization of VoIP Application Servers for Implementation of Private Unified Communication Services via LTE," In Mobilkommunikation (ITG-FB 242) Technologien und Anwendungen, Vorträge der 18. ITG-Fachtagung vom 15. bis 16. Mai 2013 in Osnabrück.
- [8] Hlaváček, J., Bešťák, R., "Improvements in the Availability of SIP Networks," In Proceedings of the 2010 Networking and Electronic Commerce Research Conference. Dallas, TX: American Telecommunications Systems Management Association Inc., 2010, pp. 109-117.
- [9] Hlaváček, J., Bešťák, R., "Live Replication of Virtualized VoIP Servers," In The Eighth International Multi-Conference on Computing in the Global Information Technology. Silicon Valley: International Academy, Research and Industry Association (IARIA), 2013, pp. 277-282.
- [10] Hlaváček, J., Bešťák, R., "Fault tolerant VoIP server based on the actor model." Submitted to Annals of Telecommunications.

- 
- [11] ITU-T Recommendation H.323 (11/96) – Packet based multimedia communication systems. <http://www.itu.int/rec/T-REC-H.323/e> [online]. November 1996.
- [12] IETF RFC 3261 SIP: Session Initiation Protocol. <http://www.ietf.org/rfc/rfc3261.txt> [online]. June 2002.
- [13] Schulzrinne, H., Rosenberg, J., “A Comparison of SIP and H.323 for Internet Telephony,” In Proceedings of NOSSDAV 1998. Cambridge, England. July 1998.
- [14] IETF RFC 3550 RTP: A Transport Protocol for Real-Time Applications <https://www.ietf.org/rfc/rfc3550.txt> [online], July 2003.
- [15] IETF RFC 33711: The Secure Real-time Transport Protocol (SRTP), <http://www.ietf.org/rfc/rfc3711.txt> [online], March 2004.
- [16] IETF RFC 3261 SIP: Session Initiation Protocol, <http://www.ietf.org/rfc/rfc3261.txt> [online], June 2002.
- [17] IETF RFC 768: User Datagram Protocol, <http://www.ietf.org/rfc/rfc768.txt> [online], August 1980.
- [18] IETF RFC 793: TRANSMISSION CONTROL PROTOCOL, <http://www.ietf.org/rfc/rfc793.txt> [online], September 1981.
- [19] IETF RFC 4960: Stream Control Transmission Protocol, <http://www.ietf.org/rfc/rfc4960.txt> [online], September 2007.
- [20] K. Singh, H. Schulzrinne, “Failover, load sharing and server architecture in SIP telephony,” *Comput. Commun.* 30, 5, March 2007, pp. 927-942.
- [21] Hlaváček, J., Bešťák, R., “Improvements in the Reliability of the Computer-Based Telecommunications Systems,” In Proceedings of Digital Technologies 2007. Zilina, Slovenska republika. November 2007.
- [22] Kelvin Chu, Ch., Pant, H., Richman, S., Wu, P., “Enterprise VoIP reliability,” In Telecommunications Network Strategy and Planning Symposium 2006. 12th International NETWORKS 2006. November 2006.
- [23] IETF RFC 5798 Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6. <http://tools.ietf.org/html/rfc5798> [online] March 2010.
- [24] IETF RFC 2281 Cisco Hot Standby Router Protocol (HSRP). <http://www.ietf.org/rfc/rfc2281.txt> [online]. March 1998.
- [25] Wu, W.; Wang, K.; Jan, R.; Huang, Ch., “A Fast Failure Detection and Failover Scheme for SIP High Availability network,” In Proceedings of the IEEE International Symposium on Pacific Rim Dependable Computing, 13th IEEE International Symposium on Pacific Rim Dependable Computing, 2007, pp. 187–190.

- 
- [26] Service Availability Forum AIS page, 2014. Application Interface Specification homepage. <http://www.saforum.org/page/16627~217404/Service-Availability-Forum-Application-Interface-Specification> (accessed June 16, 2014).
- [27] Matic, V.; Franicevic, I., Sekalec, D., “Parallel SIP Proxy Servers Using Direct Routing Approach,” In Proceedings of the IEEE Conference on Software in Telecommunications and Computer Networks, International Conference on Software in Telecommunications and Computer Networks. 2006, pp. 218-222.
- [28] Tzvetkov, V., “Fast detection of disconnection using adaptive Fuzzy Logic,” In Proceedings of the 2007 IEEE International Conference on Networking, Sensing and Control, IEEE International Conference on Networking, Sensing and Control. 2007, pp. 828-833.
- [29] Sevcik, B., Zeilinger, H., Turek, T., Zucker G., “Network Layer Based Redundancy for Time-Critical VoIP applications,” In Proceedings of the IEEE Africon conference. IEEE AFRICON, 2009, pp. 1-5.
- [30] IETF RFC 5880 Bidirectional Forwarding Detection. <http://www.ietf.org/rfc/rfc5880.txt> [online]. June 2010.
- [31] Botsch, D., Eberding, H., “A Real-Time Communication System with High-Level Language Software,” IEEE TRANSACTIONS ON COMMUNICATIONS. 1982, 30 (6), pp. 1337–1342.
- [32] Asterisk project homepage, <http://www.asterisk.org/>, [retrieved: June 2014].
- [33] Mobicents project homepage, <http://www.mobicents.org/>, [retrieved: June 2014].
- [34] JBoss project homepage, <http://www.jboss.org/>, [retrieved: June 2014].
- [35] Cisco Systems: High-Availability Solutions for SIP Enabled Voice-over-IP Networks, 2002. White paper. [http://www.cisco.com/en/US/tech/tk652/tk701/technologies\\_white\\_paper09186a00800a9818.shtml](http://www.cisco.com/en/US/tech/tk652/tk701/technologies_white_paper09186a00800a9818.shtml), [retrieved: June 2014].
- [36] Rosenberg, J., Schulzrinne, H., “Session Initiation Protocol (SIP): Locating SIP servers,” RFC3263, Internet Engineering Task Force, June 2002.
- [37] Gulbrandsen, A., Vixie, P., Esibov, L., “A DNS RR for specifying the location of services (DNS SRV),” RFC2782, Internet Engineering Task Force, February 2000.
- [38] Cisco Systems: High-Availability Solutions for SIP Enabled Voice-over-IP Networks, 2002. White paper. [http://www.cisco.com/en/US/tech/tk652/tk701/technologies\\_white\\_paper09186a00800a9818.shtml](http://www.cisco.com/en/US/tech/tk652/tk701/technologies_white_paper09186a00800a9818.shtml) [retrieved: June 2014].

- 
- [39] Leu, J., Hsieh, H., Chen, Y., Chi, Y., “Design and Implementation of a Low Cost DNS-based Load Balancing Solution for the SIP-based VoIP service,” In Proceedings of the IEEE Asia-Pacific Services Computing Conference, IEEE Asia-Pacific Services Computing Conference. 2008, pp. 310–314.
- [40] IETF RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2, <http://www.ietf.org/rfc/rfc5246.txt> [online], August 2008.
- [41] IETF RFC 5626 Managing Client-Initiated Connections in the Session Initiation Protocol (SIP). <http://www.ietf.org/rfc/rfc5626.txt> [online]. October 2009.
- [42] Gorti, A., “A fault tolerant VoIP implementation based on open standards,” In Proceedings of the IEEE Dependable Computing Conference, Dependable Computing Conference, 2006, pp. 35–38.
- [43] Hlaváček, J., Bešťák, R., “Software Architectures for High Available Telecommunication Service Platforms,” In Knowledge in Telecommunication Technologies and Optics - KTTO 2010. Ostrava: VŠB - TUO, FEI, Katedra elektroniky a telekomunikační techniky, 2010, pp. 92-96.
- [44] Kambourakis, G., et al., “High availability for SIP: Solutions and real-time measurement performance evaluation,” International Journal of Disaster Recovery and Business Continuity, vol. 1, no. 1, 2010, pp. 11-30.
- [45] Gorti, A., “A fault tolerant VoIP implementation based on open standards,” In Proceedings of the IEEE Dependable Computing Conference, Dependable Computing Conference. 2006, pp. 35–38.
- [46] Cheng, Y.; Wang, K.; Jan, R.; Chen, Ch.; Huang, Ch., “Efficient Failover and Load Balancing for dependable SIP proxy servers.” In Proceedings of the IEEE Symposium on Computers and Communications 2008, IEEE Symposium on Computers and Communications. 2008, pp. 1153–1158.
- [47] Dahlstedt, J., Vasseur, A., Bonér, J., “Java Virtual Machine support for Aspect-Oriented Programming.” 5th International Conference on Aspect-Oriented Software Development (AOSD'2006) Bonn, Germany. March 20-24, 2006.
- [48] Koutras, V. P., Platis, A. N., “VoIP Availability and Service Reliability through Software rejuvenation policies,” In Proceedings of the 2nd International Conference on Dependability of Computer Systems, 2nd International Conference on Dependability of Computer Systems, 2007, pp. 262–269.
- [49] Koutras, V. P., Salgaras, C. S., Platis, A. N., “Software Rejuvenation for Higher Levels of VoIP Availability and Mean time to failure,” In Proceeding of the 2009

- Fourth International Conference on Dependability of Computer Systems, Fourth International Conference on Dependability of Computer Systems; 2009, pp. 99–106.
- [50] Lee, M, Krishnakumar, A. S., Krishnan, P., Singh, N., Yajnik, S., “Supporting soft real-time tasks in the Xen hypervisor,” VEE 2010, Pittsburgh, PA, March, 2010, pp. 97-108.
- [51] Clark, Ch., et al., “Live migration of virtual machines,” In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, vol. 2, USENIX Association, Berkeley, CA, USA, 2005, pp. 273-286.
- [52] Scales, D. J., Nelson, M., Venkitachalam, G., “The design of a practical system for fault-tolerant virtual machines,” SIGOPS Operating Systems Review, v ol. 44, issue 4, December 2010, pp. 30-39.
- [53] Vmware homepage, VMware Incorporation, <http://www.vmware.com> [retrieved: June 2014].
- [54] Kambourakis, G., et al., “Hardware and Software Approaches for Deterministic Multi-Processor Replay of Concurrent Programs,” In Intel Technology Journal, vol. 13, issue 4, 2009, pp. 20–41.
- [55] Cully, B., et al., “Remus: High availability via asynchronous virtual machine replication,” In Proc. Symp. Network Systems Design and Implementation (NSDI), 2008, pp. 161–174.
- [56] Lee, M, Krishnakumar, A. S., Krishnan, P., Singh, N., Yajnik, S., “XenTune: Detecting Xen Scheduling Bottlenecks for Media Applications,” IEEE Globecom 2010 (Communications Software, Services and Multimedia Applications Symposium), Miami, FL, Dec 6-10, 2010, pp. 1-6.
- [57] Lee, M, Krishnakumar, A. S., Krishnan, P., Singh, N., Yajnik, S., “Supporting soft real-time tasks in the Xen hypervisor,” VEE 2010, Pittsburgh, PA, March, 2010, pp. 97-108.
- [58] Patnaik, D., Bijlani, A., Singh, V. K., “Towards high-availability for IP telephony using virtual machines,” IEEE 4th International Conference on Internet Multimedia Services Architecture and Application (IMSAA), December 2010, pp. 1-6.
- [59] Xen hypervisor homepage, <http://xen.org>, [retrieved: March, 2013].
- [60] IFB, The Intermediate Functional Block device, <http://www.linuxfoundation.org/collaborate/workgroups/networking/ifb>, [retrieved: March, 2013].
- [61] Ubuntu, Open Source Linux operating system, <http://www.ubuntu.com>, [retrieved: April, 2013].

- 
- [62] Freeswitch, Open Source Multi-Protocol Soft Switch, <http://www.freeswitch.org>, [retrieved: April, 2013].
- [63] Hlaváček, J., Bešťák, R., "Configuration of Live Migration for VoIP Applications," In Proceedings of 15th Mechatronika 2012, Praha, Czech Technical University in Prague, 2012, pp. 187-190.
- [64] Grottke, M., Trivedi, K.S., "A classification of software faults", Journal of Reliability Engineering Association of Japan, vol. 27, no. 7, 2005, pp. 425-438.
- [65] Raymond, E. S., The New Hacker's Dictionary. The MIT Press, Cambridge, 1991.
- [66] Statistical Analysis of Floating Point Flaw: Intel White Paper, <http://www.intel.com/support/processors/pentium/sb/CS-013017.htm> [retrieved: June 2014].
- [67] Huang, Y., Kintala, C., Kolettis, N., Fulton, N.D., "Software rejuvenation: analysis, module and applications," Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., June 1995, pp.3 81,390, 27-30.
- [68] Alonso, J., Bovenzi, A., Jinghui Li, Yakun Wang, Russo, S., Trivedi, K., "Software Rejuvenation: Do IT & Telco Industries Use It?," Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on, 2012, pp. 299.
- [69] Hewitt, C., Bishop, P., Steiger, R., "A Universal Modular Actor Formalism for Artificial Intelligence," IJCAI, 1973.
- [70] Armstrong, J., Erlang. Commun. ACM 53, 9, September 2010, pp. 68-75.
- [71] Akka project homepage, <http://akka.io/>, [retrieved: July 2014].
- [72] Java language homepage, <https://www.java.com/>, [retrieved: July 2014].
- [73] Scala language homepage, <http://scala-lang.org/>, [retrieved: July 2014].
- [74] Koutras, V. P.; Platis, AN., "VoIP Availability and Service Reliability through Software Rejuvenation Policies," Dependability of Computer Systems, DepCoS-RELCOMEX '07, June 2007, pp.262,269.
- [75] Sharma, V.S., Trivedi, K.S., "Reliability and Performance of Component Based Software Systems with Restarts, Retries, Reboots and Repairs", Software Reliability Engineering, 2006. ISSRE '06. 17th International Symposium on, pp. 299.
- [76] Vinayak, R., Dharmaraja, S., "Semi-Markov Modeling Approach for Deteriorating Systems with Preventive Maintenance," International Journal of Performability Engineering, Volume 8, Number 5, September 2012, Paper 6, pp. 515-526.
- [77] Sirjani, M., Movaghar, A., Shali, A., de Boer, F.S., "Modeling and verification of reactive systems using Rebeca," Fundam. Inf. 63(4), 2004, pp. 385-410.

- 
- [78] Rebeca language homepage, <http://www.rebeca-lang.org/wiki/pmwiki.php>. [retrieved: July 2014].
- [79] Vinoski, S., "Reliability with Erlang", *Internet Computing*, IEEE, vol. 11, no. 6, 2007, pp. 79.
- [80] Charousset, D., Schmidt, T. C., Hiesgen, R. and Wählisch, M., "Native actors: a scalable software platform for distributed, heterogeneous environments," In *Proceedings of the 2013 workshop on Programming based on actors, agents, and decentralized control (AGERE! '13)*. ACM, New York, NY, USA, pp. 87-96.
- [81] Apache license homepage, <http://www.apache.org/licenses/LICENSE-2.0.html>, [retrieved: July 2014].
- [82] Dijkstra, Edsger W., "On the role of scientific thought". *Selected writings on Computing: A Personal Perspective*. New York, NY, USA: Springer-Verlag. 1982, pp. 60–66.
- [83] Chechina, N., Trinder, P., Ghaffari A., Green, R., Lundin, K. and Viriding. R., "The Design of Scalable Distributed Erlang," In the *Draft Proceedings of the Symposium on Implementation and Application of Functional Languages 2012 (IFL'12)*, Oxford, UK, 2012.
- [84] Typesafe homepage, <http://typesafe.com/>, [retrieved: July 2014].
- [85] Armstrong, J. L., "Erlang-an experimental telephony programming language," *SR Viriding - XIII International Switching Symposium*, 1990.
- [86] Jian, S., Shaoping W., Yaoxing, W., "Petri-nets Based Availability Model of Fault-Tolerant Server System," *Robotics, Automation and Mechatronics, 2008 IEEE Conference on*, pp. 444.
- [87] Mullen, R.E. 1998, "The lognormal distribution of software failure rates: origin and evidence," *Software Reliability Engineering, 1998. Proceedings. The Ninth International Symposium on*, pp. 124.
- [88] Rahmani, C., Srinivasan, S.M. and Azadmanesh, A., "Exploratory failure analysis of open source software," *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, 2010, pp. V1-51.
- [89] TimeNet 4.0 homepage, <http://www.tu-ilmenau.de/sse/timenet/>, [retrieved June 2014].
- [90] Salfner, F., Wolter, K., "A Petri net model for service availability in redundant computing systems," *Simulation Conference (WSC), Proceedings of the 2009 Winter, 2009*, pp. 819.
- [91] Kamon project homepage, <http://www.kamon.io>, [retrieved: July 2014].