A P2PSIP event notification architecture

GEORGIOS PANAGIOTOU

KTH Electrical Engineering

Master's Degree Project
Stockholm, Sweden 2008

To be given by the department

A P2PSIP event notification architecture

Georgios Panagiotou

October 3, 2008

Examiner
Viktoria Fodor

Industrial Supervisor
Alisa Devlic, Appear Networks AB

Department of Electrical Engineering
Royal Institute of Technology (KTH)
Stockholm, Sweden

Abstract

This M.Sc. thesis, part of the EU FP6 IST project MUSIC, presents a P2PSIP event notification architecture where a traditional SIP event server is replaced by a distributed P2P overlay network. Our objective is a scalable, fault resilient, and redundant SIP architecture that meets the state of the art in handling event state subscriptions, notifications, and publications.

The users of the architecture - non overlay SIP user agents - continue to transparently access the overlay according to the IETF specifications for event state publications (RFC3903 [1]), and notifications (RFC3265 [2]). The proposed architecture leverages a structured P2P overlay network, an application level multicast protocol, and a modified SIP stack that invokes the P2P functionality. The overlay installs a routing and transport infrastructure that distributes the total processing load among the peers. The application level multicast protocol builds dissemination trees each of which serves a particular subscriber group. It also provisions for the failure and recovery features of the overlay in coordination with the P2P subsystem. Finally the SIP stack receives out of overlay SIP calls and invokes the internal P2P functionality. However, not all the overlay peers need to have a SIP stack; only the ones acting as gateways to out of overlay SIP requests. This design allows for lightweight peers, completely unaware of SIP operations and semantics.

Moreover, we are evaluating the performance of the notification mechanism in a mobile environment where the peers disconnect and join the overlay at random intervals. The evaluation is performed with a packet level simulator that introduces a variable churn rate to the P2P overlay network. The simulation results show that the architecture experiences service discontinuities when the overlay connections break faster than the overlay repairs these connections. However it is also seen that, when appropriately configured, the notification architecture can partially mitigate the effect of churn and achieve a stable operation.

Acknowledgements

This thesis project concludes my M.Sc. studies in KTH, The Royal Institute of Technology, with a major in Electrical Engineering - Wireless Systems. In this Section of the thesis I would like to acknowledge certain persons, that without their support, I would have never reached this point. Primarily, it is my parents that helped and supported me the most, both economically and mentally, during my studies. Furthermore, I would like to thank all the people in Appear Networks and especially Alisa Devlic that actively supervised the thesis work, and collaborated in many parts of it. Many thanks also to Viktoria Fodor, my examiner from KTH, The Royal Institute of Technology, that always provided valuable feedback and catered for the punctual completion of the thesis work, even more than myself. Last but not least, are my friends Carlos Angeles, John Glaropoulos, Andreas Strikos, Thanos Makris, Chris Papazafeiropoulos, and a few more, that with their unique personality influenced either directly or indirectly this thesis project.

Contents

List of Tables

List of Figures

Abbreviations

3GPP	3d Generation Partnership Project
ALM	Application Level Multicast
API	Application Programming Interface
CAN	Content Addressable Network
CIDR	Classless Inter-Domain Routing
DHT	Distributed Hash Table
FPF	Forward Path Forwarding
HIP	Host Identity Protocol
ICE	Interactive Connectivity Establishment
IETF	Internet Engineering Task Force
IP	Internet Protocol
IST	Information Society Technology
KBR	Key Based Routing
MADAM	Middleware for Adaptive Applications
MIDAS	Middleware for the Deployment of Mobile Services in Ad-Hoc Networks
MUSIC	Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environment
NAT	Network Address Translator
P2P	Peer to Peer
PNS	Proximity Neighbor Selection
POTS	Plain Old Telephone Service
RDP	Relative Delay Penalty
RP	Rendezvous Point
RPF	Reverse Path Forwarding
RTT	Round Trip Time
saScribe	SIP aware Scribe
SHA	Secure Hash Algorithm
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIMS	Semantic Interfaces for Mobile Services
SIP	Session Initiation Protocol
SOA	Service Oriented Architecture
SPS	SIP Presence Server
STUN	Simple Traversal of User datagram protocol (UDP) through Network address translators (NATs)
TURN	Traversal Using Relay NAT
URI	Uniform Resource Identifier

Chapter 1
Introduction

As the IT industry evolves, end nodes tend to gain more and more efficiency. Nowadays, small in size, yet powerful in computational resources, mobile terminals are becoming a trend. These devices are often called “smart” because of their advanced characteristics that result in an enhanced user’s experience. Furthermore, as the industry is moving towards the 4^{th} generation (4G) networks, these devices gain advanced interconnection capabilities that implement the vision for a best and always connected paradigm. Now that technology can provide the conditions for a holistic improvement in the user’s experience, advanced applications need to be implemented that both satisfy the user demands and exploit the underlying technological infrastructure.

However, special requirements arise when designing such applications. More specifically, in a mobile context where the execution environment changes rapidly the deployed applications need to adapt their operation according to the users’ needs and situation. These class of applications were first described by Schilit et al. [6] as context aware applications. The term context often cause disambiguation and this report adopts the representative definition given by Dey and Abowd [7]. According to them, context can be considered as "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". Modern context-aware applications exploit the context information of an entity in order to adapt their behavior, thus enhancing the user’s experience.

Traditionally, these applications leverage context information accessed through the user’s device (i.e., calendar information, device sensors, etc.). This turns out to be a severe limitation as the user’s environment often includes information that is not directly accessible from the device and needs to be remotely communicated. Thus, global context acquisition and information knowledge is the key for true context-aware applications. A joined research effort with both academic and industrial partners in the domain of contemporary context-aware systems is the MUSIC project. MUSIC stands for Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environment and is a research project funded by the European Commission under the Information Society Technology (IST) priority under the 6^{th} Framework Program. The MUSIC project provisions an open platform for developing applications capable of adapting in highly dynamic user and execution contexts, while maintaining a high level of usefulness across context changes. One of the key features in the MUSIC context architecture is the distribution of context information.

Disseminating context information regardless of where it is produced invokes many technical challenges, since the design of a robust and efficient transport scheme able to operate over heterogeneous networking environments is by no means a trivial task. The designer of such a context distribution architecture has to cater for a plethora of requirements in order to create a context distribution architecture that is scalable, robust and capable of transferring volatile context information fast and efficiently. The requirements that a context distribution method has to meet are explained below and are given in brief on Table 1.1.

Unified addressing scheme:
The distribution method needs an addressing scheme completely decoupled from the underlying network technologies [8]. That feature enables for context transfer over heterogeneous networking technologies that use different network level identifiers (i.e IPv4 and IPv6 addresses for IP networks, VPI/VCI identifiers for ATM networks, etc.). However, even when context distribution occus over a single networking technology a unified addressing scheme provides an elegant solution to the problem of multihoming [9].

Aggregation:
When a context aware application accesses information from many remote context sources an aggregation mechanism could save computation and network resources. In turns an efficient resource management strategy results in more scalable context transfers.

State of the art context distribution:
A contemporary context distribution mechanism should enable for both synchronous (pull-based) and asynchronous (push-based) context retrieval. In a pull-based model often so called request-response the context consumer issues context requests as soon as it needs access to the context information. On the contrary in a pull-based model often so called publish-subscribe, context information exchange occur upon changes in the information [10].

Deployment:
The distribution method must utilize technologies that either are or will be deployed, ensuring interoperability between systems.

Scalability:
The solution should ideally provide Internet-wide scalability.

Robustness:
The distribution method should provision for networking environments with limited or no connectivity.

Quick distribution:
As context information is often volatile, the distribution method should be quick enough to deliver context information on time without excessive delays that may result in outdated context information.

Economically viable:
The distribution method should integrate smoothly with the context-aware system in order to be eligible for quick adaptation and deployment.

*	Unified addressing scheme
*	Aggregation support
*	State of the art context distribution
*	Deployment
*	Scalability
*	Robustness
*	Quick distribution
*	Economically viable

Table 1.1: Summary of context distribution requirements

The context distribution architecture currently adopted by the MUSIC project is based on the event notification mechanisms of the Session Initiation Protocol (SIP). This architecture is mainly inspired by the research conducted in [11], where SIP and SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions) are utilized towards an efficient and robust transport infrastructure for exchanging context information. However, the main concern when leveraging SIP/SIMPLE to transfer context information is the centralized design of the solution. The proposed architecture in [11] is server-oriented and as a result any generated information is destined to, or originated from a globally accessible entity, the SIP Express Router. However, centralized architectures impose certain limitations especially when it comes to scaling, and single point of failure issues; issues that decentralized and distributed architectures handle in an elegant and efficient manner.

This thesis presents a decentralized event notification architecture intended to be used by the MUSIC project, as a context exchange mechanism. The proposed notification scheme utilizes off-the-shelf technologies such as peer-to-peer (P2P) networks and application level multicast (ALM) to distribute context information among MUSIC-capable devices. The proposed architecture contributes to the P2P-SIP efforts by extending the SIP/SIMPLE presence package in a way to support for decentralized event notifications. In the advantages of the proposed architecture is the provided transparency to client applications that continue to access the overlay as they would in a centralized - traditional - SIP/SIMPLE architecture.

The rest of this report is structured as follows: Chapter 1 provides the introduction to this thesis, gives the problem statement through a motivating scenario, and also the research methodology followed. Chapter 2 elaborates on the requirements of the desired distributed and decentralized context dissemination architecture, where Chapter 3 presents the proposed dissemination architecture in detail. The evaluation of the proposed notification architecture is given in Chapter 4. Finally, Chapter 5 draws the conclusions and sets the guidelines for future work in the area.

1.1 The MUSIC project

Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environment (MUSIC) is an integrated project founded by the European Commission under the Information Society Technology (IST) priority. The main objective of MUSIC is to develop an open technology platform for software developers in order to facilitate the development of context aware and self-adapting mobile applications. The MUSIC platform provides the enabled applications a transparency layer in order to be fully operational regardless of the context change rate [12]. By that way applications are able to dynamically adapt their functionality (self-adaptation) and re-arrange their operation according to context changes.

The MUSIC platform follows a layered architecture [13], which is depicted in Fig.1.1. Initially, the architecture is divided into two main blocks: the MUSIC Studio and the runtime environment. The MUSIC Studio provides the required tools and software libraries that enables the developers build context-aware applications. These tools are essential for linking the applications with the MUSIC platform. On the other hand the runtime environment block is composed from three different layers: the System Services, the Middleware, and the Applications.

The Application layer can be seen as an interface between third party MUSIC-enabled applications and the core of the MUSIC middleware engine. The Middleware layer is further divided into the context middleware

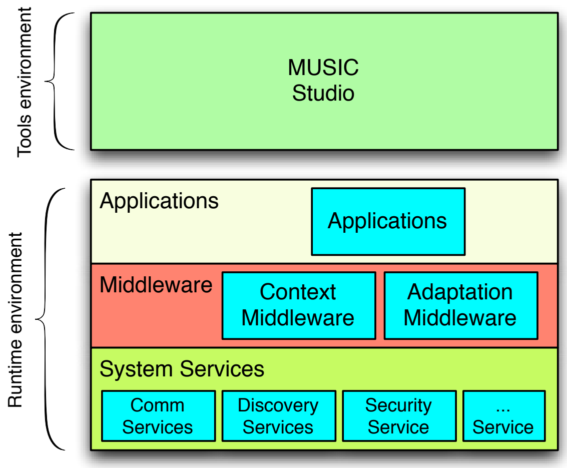


Figure 1.1: The MUSIC architecture

and the adaptation middleware. The context middleware collects, organizes, manages, and shares context information, thus making this information available to the adaptation middleware. Furthermore, the context middleware may be also used directly by context-aware applications. The adaptation middleware analyzes the changes of the context and its impact on the application(s) in order to adapt the set of running applications to the current circumstances [13].

The context middleware is further divided into several components, as shown in Fig.1.2. The Context Manager is the root hub of the context middleware and provides interoperability and coordination among the rest of the components. The MUSIC middleware interfaces various context sensors through the Context Providers Manager in order to access locally generated context information. The gathered context is then stored by the middleware for fast and efficient retrieval by the MUSIC-enabled applications. The storage functionality of the middleware is provided by the Context DB component. Context DB is designed to handle volatile and non-volatile context information in an efficient way, since it implements both a context cache and a context database. The context cache stores context information that is frequently accessed, where context database stores context information that is valid for longer time periods. Applications issue context requests encoded in the Context Query Language (CQL), which resembles the SQL in structure. The component that processes and manages the context queries coming from the applications is the Context Query Processor. Additionally, the Ontology Manager manages the collection of ontologies which are available by the given instance of the middleware. Finally, the component that decouples the middleware from accessing only local context information is the Context Distribution Manager. Context Distribution Manager communicates with the Context Distribution Service, which implements the required transport infrastructure for efficiently disseminating context information among various devices and instances of the MUSIC middleware, thus enabling the ubiquitous computing paradigm. In an ideal deployment, the Distribution Manager component should be able to transfer context information choosing among a variety of different Distribution Service realizations, that distribute context using different transport mechanisms (i.e., SIP, HTTP, XMPP, etc.), each with their own unique advantages and characteristics.

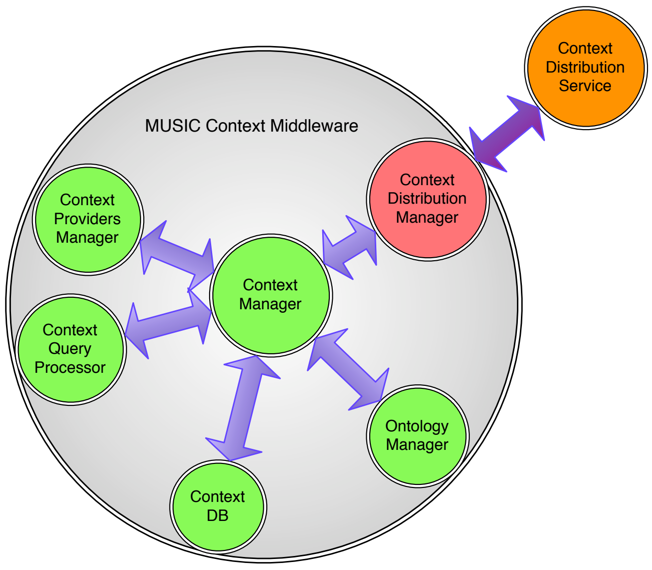


Figure 1.2: The components comprising the MUSIC middleware architecture

Currently the design of the context distribution service component is utilizing IETF's SIP/SIMPLE specifications to disseminate context information. However, since the distribution service is not part of the context middleware it has to be modeled as a general purpose distribution library, able to transfer every kind of information. Fig.1.3 shows the proposed distribution architecture currently adopted by the MUSIC project, where context information produced on the Node A is to be consumed by a context-aware application residing on Node B. The distribution plugin should provide the actions of resource location and context distribution. With resource location a middleware instance becomes aware of the context information exported by other middleware instances; where context distribution caters for the distribution of context information between different middleware instances.

Under the SIP/SIMPLE implementation of the distribution service, every context sensor is assigned a SIP Uniform Resource Identifier (URI), coupled with the sensor's metadata information. According to Fig.1.3, the resulting SIP URI for the web camera sensor on Node A can be in the form of: *sip:web.camera.sensor@sip-domain.org*. As soon as the distribution service on Node A is notified for the presence of the web camera sensor by the context middleware, it forms the associated SIP URI for that metadata and exports it to the SIP presence server through the aid of the SIP REGISTER [14] message. The SIP REGISTER message registers a SIP Address of Record (AoR) URI with one or more contact URIs. In our case the AoR is the metadata SIP URI, where the contact URI is the host's SIP URI (i.e. *sip:node.a@sip-domain.org*). Thus the REGISTER message notifies the SIP presence server, that the communicated AoR is available by the associated contact URI. In case more than one middleware instances export the same AoR (i.e. they have the same sensor plug-in installed) the SIP presence server maintain a URI contact list associated to that AoR.

Following the REGISTER strategy, it is possible to communicate the plugins' metadata available at every instance of the middleware at a centralized location, the SIP presence server. When a node wishes to acquire

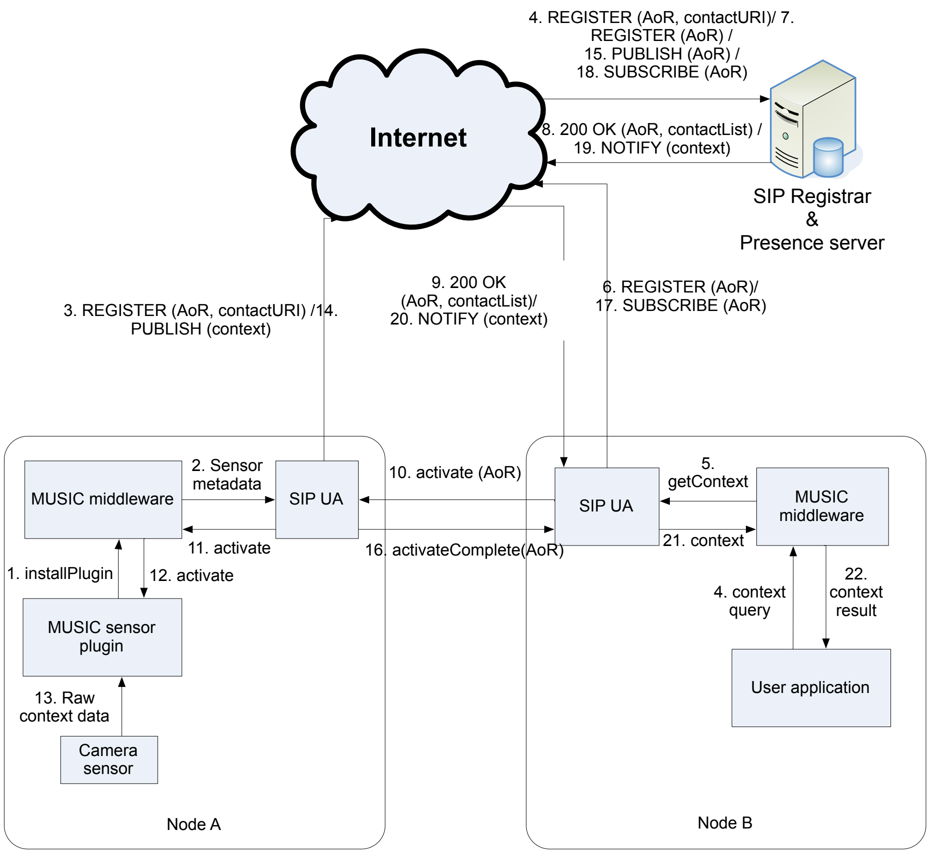

4. REGISTER (AoR, contactURI)/ 7. REGISTER (AoR) / 15. PUBLISH (AoR) / 18. SUBSCRIBE (AoR)
Internet
8. 200 OK (AoR, contactList) / 19. NOTIFY (context)
SIP Registrar & Presence server
3. REGISTER (AoR, contactURI) /14. PUBLISH (context)
9. 200 OK (AoR, contactList)/ 20. NOTIFY (context)
6. REGISTER (AoR)/ 17. SUBSCRIBE (AoR)
MUSIC middleware
2. Sensor metadata
SIP UA
10. activate (AoR)
SIP UA
5. getContext
MUSIC middleware
11. activate
16. activateComplete(AoR)
21. context
1. installPlugin
12. activate
4. context query
22. context result
MUSIC sensor plugin
User application
13. Raw context data
Camera sensor
Node A
Node B

Figure 1.3: The Distribution Service architecture currently proposed in the MUSIC project

the list of the nodes offering a particular service it simply issues a special crafted SIP REGISTER message to the presence server requesting the contact list for that particular AoR. In the example presented by Fig.1.3, Node B issues a SIP REGISTER message in query mode destined to the presence server, requesting every available information regarding the AoR of the web camera sensor. In that case the server responds that the requested AoR is provided by Node A, and the resource location functionality is over.

After Node B having met the resource location objective, initiates the context distribution functionality provided by the distribution service component. For distributing the context information, a two-stage architecture is employed, where the sensor providing the desired context information has to be first activated (if not already), in order to start disseminating the context information. In the proposed distribution service the activation procedure is implemented with the aid of the SIP MESSAGE message, intended to be used directly between two SIP nodes. Thus, Node B sends an activate command, encapsulated into a SIP MESSAGE message, destined to the provider of the desired context information, the Node A. As soon as Node A receives the activation command, activates the appropriate context sensor and starts to receive context information which publishes to the presence server. Node B subscribes at the presence server using as identifier the AoR of the camera sensor, and gets the context information notifications. The subscription and notification procedures are handled by the SIP PUBLISH [1], and SUBSCRIBE/NOTIFY [2] messages, and are thoroughly explained later on.

The main advantage of the currently proposed distribution architecture is its simple and robust design that installs custom functionality only at the end hosts, while the operations at the presence server are kept fully compliant with the IETF RFCs. Since no custom functionality is required by the presence server, the proposed architecture can be quickly prototyped, demonstrated and tested. On the other hand, every generated request or response for context information is destined to, or originated from the presence server. The current decision for a centralized design creates concerns regarding the scalability and fault tolerance of the architecture. Relevant research conducted in [11] proofed that scaling is not an issue for the architecture, since it is able to handle a large number of subscribers. However, there are open issues regarding the robustness and fault tolerance of this centralized approach. To the best of our knowledge, centralized architectures often implement over-provisioning in order to tackle these issues; it is common truth though that the tradeoff for over-provisioning is the increased implementation and operational cost that drastically reduces the applicability of the solution.

1.2 Motivation scenario

In this section we derive a motivation scenario that shows the advantages of a decentralized and distributed context dissemination architecture. Technology assisted emergency scenarios are commonly considered as realizations of practical use cases. Moreover, emergency scenarios are used worldwide in order to minimize the effects of a real emergency situation, particularly in respect of saving human lives and restricting collateral losses. The following scenario is inspired by emergency processes compiled by the MUSIC [12] project consortium, for the major Paris transit operator RATP.

The emergency scenario described here, takes place in the Paris metro and the main concept is that a metro train departs from the station “Gare de Lyon” heading towards station “Nation”. These two stations are part of the Paris metro red line and both are part of the inner Paris metro zone (1st zone) [15]. While the train approaches “Nation” a sudden unexpected and unknown mechanical failure occurs and the metro train stops.

At this point we consider an advanced inter-train infrastructure for efficiently handling emergency situations. The scenario assumes that each metro train provides telemetry information to both ends of its current route. In this scenario these stations are “Gare de Lyon” and “Nation”. For accompanying the needs of the scenario, telemetry information includes any train generated data (i.e. speed, exact position on the track, etc.) that are transmitted to the train stations, and to the RATP Traffic Control (RTC) center. With that requirement the damage and the current status of the train can be evaluated and counteract actions can be employed. It is further assumed that the train operator is equipped with a PDA running the MUSIC context middleware. The PDA is able to communicate with the respective stations and the RTC through the same track-line infrastructure. The infrastructure that communicates any train / operator generated information to the stations and vice versa, is rather simple and can be seen as a traditional bus-artery that hosts a series of wireless bridges / repeaters (Fig.1.4). As soon as the transmitted information is captured by one of the wireless transceivers deployed along the track, it is communicated to the respective stations.

The emergency scenario specifies that the train suddenly fails in the middle of the track prior to station “Nation”, and some of the passengers might get injured due to the abrupt stop. Train electronics and sensors detect the problem and notify the corresponding stations along the track that signal the RTC center. Both

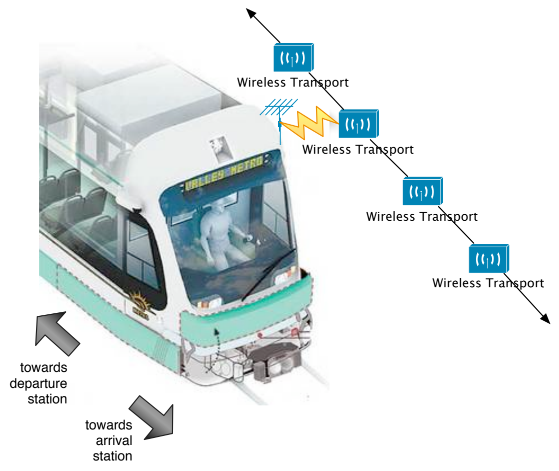


Figure 1.4: The track infrastructure proposed by the derived emergency scenario

the local train stations and the RTC have to efficiently collect any available context information from the sensors in the incidents place, combine them, and efficiently distribute the stations resources. Collecting the right context information at the right time results in minimizing damages and re-engage the normal operation of the train stations as soon as possible. Fig.1.5 depicts the interworking between the various entities defined by the emergency scenario. According to that figure, as soon as the train fails, the telemetry information is propagated along the track infrastructure to the stations. Through a WAN technology the stations further communicate the information to the RTC. The operator of the train is able to establish independent communication with the stations and the RTC, utilizing its PDA. The train passengers are informed about the imminent actions regarding the re-establishement of the train's operation through an audio/visual system that displays operator updates.

Additionally to their information forwarding operations (between the train and the RTC) the stations manage a pool of available resources. These resources mainly consist of station personnel (i.e. engineers, doctors, paramedics), ready to be disposed at the incidents scene. However, in case the situation demands it third party resources might be engaged as well (i.e. fire brigade, police, ambulances). In addition to the units heading towards the failed metro train, additional resources have to regulate the situation in both stations. For example, station's security personnel have to be engaged, information should be given to the passengers waiting at the docks, and in cooperation with the RTC an estimation of the track recovery time should be calculated. If the time needed to bring the track back to operation exceeds certain limits, alternative means of transportation (i.e. busses, tram) have to be provided to the station passengers. In the described scenario every entity either physical (i.e. train operator) or logical (i.e. RTC) that participates in the incident is equipped with MUSIC enabled devices. That is, the MUSIC context middleware instance is running on each of these devices, providing efficient context aware capabilities. Elaborating the entire use case invokes many parameters that have to be considered. Therefore, we limit our approach on the failed train and its immediate surrounding. More specifically we are interested in the communication channel that has to be set-up between the metro train, train's operator, as well as, between the station and third-party personnel.

The MUSIC project, in it's current version, adopts SIP/SIMPLE as the transport framework for efficient

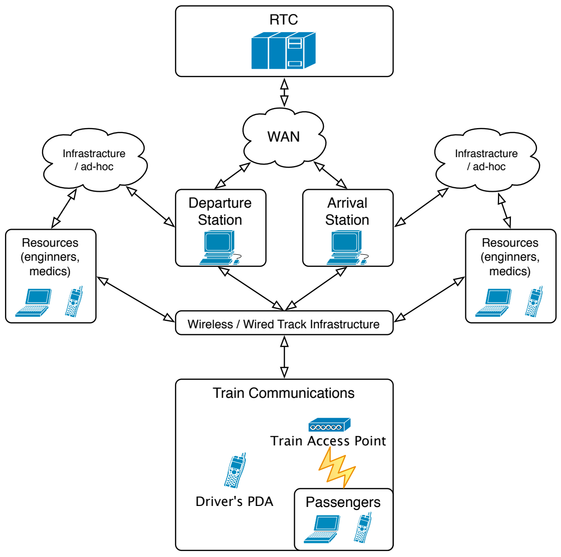


Figure 1.5: Interworking in the emergency scenario

context distribution. However, the SIP architecture operates on a client-server basis, maintaining a single point of contact, the SIP presence server (SPS). As analyzed in [11], the main objective of SPS is to collect context information from the producers and distribute it to the registered context consumers. Utilizing SIP/SIMPLE as the context transport engine in the derived emergency scenario, invokes several pitfalls. Mainly, in terms of single point of failure and communication latency. Maintaining a single point of contact for the context exchange may result in a total communication breakdown, in case the SPS becomes unavailable (i.e., network or equipment failure). Additionally, context information that is produced and consumed at the incidents place can experience increased delays when routed through the SPS, acting as a broker. The on time delivery of context information is very important for particular scenarios, like the one we have described. Moreover, stale context info utilizes network and computational resources without providing any value.

In this thesis the main objective is to replace the classical server-oriented distribution architecture with a distributed and decentralized equivalent. The rest of this report elaborates on the architectural details of this technical challenge. For the scenario compiled in this section, it would be ideal if the context information produced in the area of the emergency should be communicated to all the different entities through a common communication channel. Modeling the dissemination architecture with a common ring yields faster context distribution times, as topologically closer entities can be placed adjacently on the ring. In addition. the problem of keeping a single context dissemination point can be alleviated accordingly, when an efficient dissemination strategy is applied. Moreover, a distributed context dissemination architecture results in an economy of scale when compared to an over-provisioned centralized scheme, as the one proposed in [11]. Finally, Fig.1.6 depicts our vision on a context distribution scheme that inherits the primitive of a common communication ring.

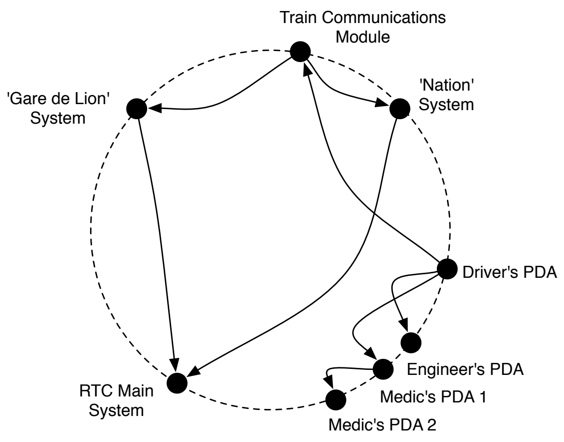


Figure 1.6: Disseminating context information through a common communication channel

1.3 Longer problem statement

In this thesis work we propose a context distribution architecture to be used by the MUSIC project. According to the introduction the derived architecture has to meet certain requirements in order to be eligible for deployment. This thesis focuses on the design of a fully distributed and decentralized context dissemination scheme, that eventually should be able to compete against centralized equivalent designs. Section 1.1 gave an insight into the context distribution service proposed by the MUSIC consortium, that utilizes SIP/SIMPLE for context delivery. However the service architecture proposed raises scalability and fault-tolerance issues. On the contrary, a competing context dissemination service of distributed and decentralized nature appears appealing characteristics, since it provides elegant solutions to these issues of centralized nature.

The emergency scenario described in Section 1.2 sets the vision for a decentralized and distributed context distribution scheme. According to the derived scenario a common communication ring could be utilized to efficiently distribute context information. Context consumers and producers simply join the ring in order to have access to the generated information. Modeling the communication channel as a multiple access ring is not an arbitrary choice, since it resembles the architecture adopted by structured peer-to-peer (P2P) networks. The peers participating in a structured P2P network can easily share and retrieve context information on demand. In this thesis it is proposed that the abstract communication channel of the motivation scenario should be implemented with a P2P technology. However a P2P network overlay only installs the routing infrastructure. That is, given a key the P2P overlay routes that key to the peer responsible for that key. For efficiently disseminating context information within the overlay a smart context distribution mechanism has to be employed. Such distribution mechanisms can be implemented utilizing application level multicast (ALM) technologies.

Combining off-the-shelf technologies like P2P and ALM in order to implement a distributed and decentralized context distribution network, like the one described in Section 1.2, is not a trivial task. Mainly because these technologies were primarily designed for use in fixed hosts and the robustness of a P2P architecture is questionable when the operating environment introduces terminal mobility. Our intension is to develop a context distribution architecture intended to run efficiently over every network, wired or wireless.

This requirement for ubiquitous service provisioning sets challenges to the selected technologies. Briefly, peer churn is the term used in P2P terminology for defining the situation where peers join and leave the P2P network at random. This randomness as we will discuss in Section 2.2, destabilizes the P2P structure and certain procedures have to be performed such that the P2P network can be fully operational again. While the overlay is destabilized the ALM solution that uses the P2P routing functionality is also affected. Under high churn load the P2P system can break leading to a total communication breakdown. Primary objective for this thesis project is to investigate how these technologies respond to unstable networking environments and to propose applicable modifications towards their potential adaptation as the context distribution service of the MUSIC project.

Summarizing the requirements for the architecture, it has to enable the resource location and context distribution. Efficient resource location can be provided with the aid of a structured P2P technology, where scalable and robust context distribution can be achieved with a smart multicast scheme. Moreover, it is desirable that the derived architecture should be able to operate completely decoupled from the MUSIC project, as a general purpose transport library with distributed and decentralized features. Towards that direction, one way to push for faster adaptation and deployment is to provide coupling with the SIP/SIMPLE event notification package. Thus, the applications that already utilize SIP/SIMPLE for disseminating events (i.e. instant messaging applications), could enable distributed and decentralized features with relatively few changes in their software design.

1.4 Methodology and tools

For the evaluation of our proposed architecture we utilize a discrete packet-level simulator. The selected simulator is Oversim [3], since it is specially developed for the simulation and evaluation of P2P systems. Oversim is not a standalone simulation framework, yet it is available as an extension of the well known simulator Omnet++ [16]. In Fig.1.7 the Oversim modular architecture is depicted. According to the figure, Oversim supports a plethora of structured P2P networks as well as some technologies (i.e. application level multicast) that exploit the underlying P2P systems. Using Oversim it is possible to acquire measurements, without having to develop and test from scratch a custom simulator.

This thesis proposes a context distribution architecture based on P2P systems and ALM technologies, to support efficient, scalable, and robust resource location and context distribution. Using Oversim, we can test a context dissemination scheme that resembles the proposed architecture in characteristics and behavior. We test the architecture in an environment of dynamic membership, where peers join and leave the overlay at random. Testing the architecture in a rapidly changing networking environment provides proof-of-concept regarding the applicability of the architecture in a real deployment.

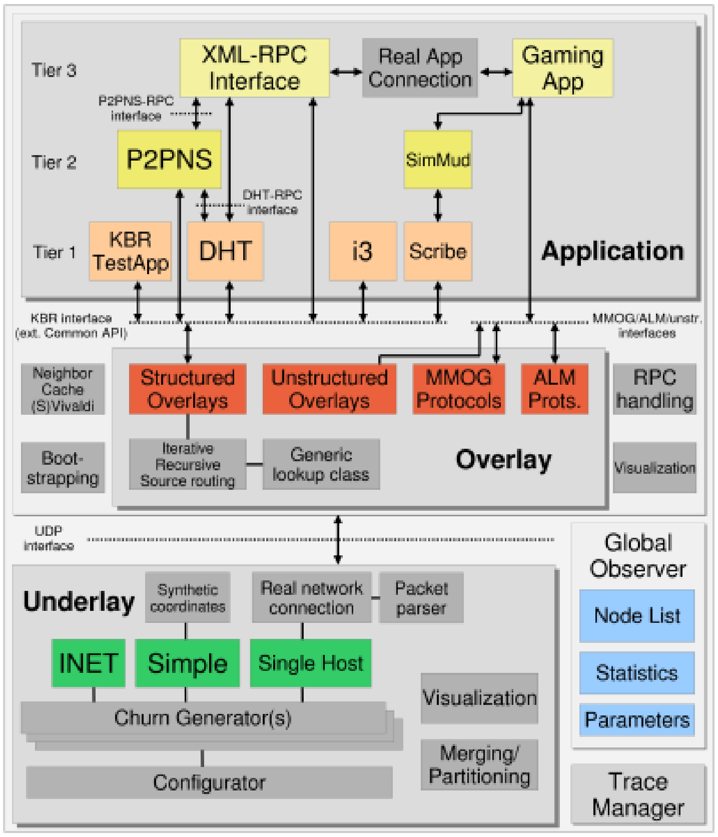


Figure 1.7: The Oversim architecture [3]

Chapter 2
Background

2.1 Event notification services

Event notification services provide an elegant and attractive solution for the communication of events between interested entities. Within the context of this thesis these entities are the various context consumers and producers participating in a MUSIC domain. A contemporary event notification service actively supports synchronous and asynchronous event communication. Synchronous communication is realized with a legacy request-response, pull-based, system, where the event consumers regularly query the producers for notifications. On the other hand, asynchronous communication is usually realized with a publish-subscribe, push-based, system. Publish-subscribe systems gain more and more attention since their use in wide-area networks, with first case the Internet, incorporates many advantages. A publish-subscribe system operates in a completely decoupled fashion, letting the producers publish their events into the system while being completely unaware of the consumers. A publish-subscribe system can be further categorized into topic-based and content-based, according to the different techniques the subscriber can employ to access the published events.

In a topic-based publish-subscribe system both the publishers and the subscribers share a common identifier (i.e. group name) that uniquely identifies the topic of interest. The subscribers use the identifier to get access to the published information, and the publishers to publish their information. A well-known application that uses topic-based functionality is a mailing list, where a single email destined to the mailing list is received by every subscriber. On the other hand, in a content-based publish-subscribe approach as soon as a subscriber enters the system it provides its content profile. For every generated event the system performs a predicate matching on the subscriber's profile, it follows that the event is communicated to the subscriber only if it matches his/her content profile. Clearly, the challenge in the case of content-based systems is the design of a matching algorithm that scales well as the volume of subscribers in the system massively increases.

2.1.1 SIP / SIMPLE

The Session Initiation Protocol (SIP) [14], among other, specifies a topic-based publish-subscribe architecture. SIP is a general purpose signaling protocol with open specifications standardized by the Internet Engineering Task Force (IETF) initiative. SIP incorporates an expandable architecture where new features can be added in order to accommodate arising demands; thus non-IETF bodies have also adopted SIP for communicating control information of interest (i.e. 3GPP). Leveraging SIP as the signaling protocol for an event notification service employs many advantages primarily because SIP is a mature protocol, actively deployed and supported by various devices. Furthermore, SIP's operation is completely decoupled from applications and independent of any underlying transport protocol, allowing for session mobility over different access networks as well as between applications on different devices. SIP enables for user mobility as well, since it utilizes an abstract addressing scheme other than IP (SIP Uniform Resource Identifiers (URIs) [17]) that can uniquely address users regardless of the device they are operating. Finally, SIP's NAT penetration capabilities, supporting the ICE [18] framework with STUN [19] and TURN [20], made it ideal for use by applications that raise demands for real-world communications.

SIP defines the SIMPLE [21, 22] (SIP for Instant Messaging and Presence Leveraging Extensions) package to support scalable and efficient event notification services. The SIMPLE package actively supports both synchronous and asynchronous information retrieval mechanisms. SIP/SIMPLE leverages the PUBLISH (RFC3903 [1]) and SUBSCRIBE/NOTIFY (RFC3265 [2]) messages to enable synchronous and asynchronous event notifications. According to Fig. 2.1 the event notification model of SIP is primarily synchronous and optionally asynchronous. On the reception of a valid SIP SUBSCRIBE (M3-M4) the SIP server has to immediately reply the latest known event state (M5-M6 *synchronous* behavior). In addition, should the event state changes (i.e. through a valid PUBLISH request, M1-M2 or M7-M8) and the subscription is still active, the server should communicate the event change as a new SIP NOTIFY request back to the subscriber (M9-M10 *asynchronous* behavior).

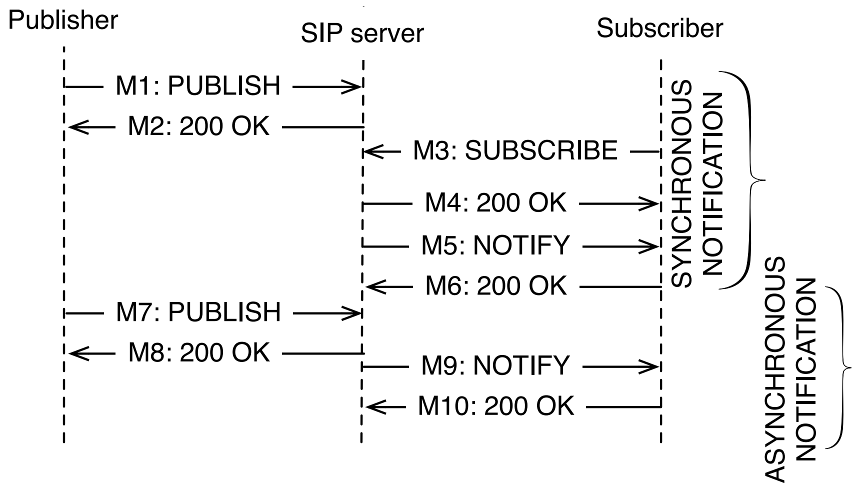


Figure 2.1: SIP/SIMPLE support for synchronous and asynchronous event notifications

The life cycle of a subscription is defined by the value of the SIP *Expires* header that is negotiated between the SIP user agent and server at the subscription set-up phase (M3-M4). A zero expiration value results in a behavior that is synchronous and state-less, as the server immediately replies the latest known event state and releases any transaction specific state. Consecutively, an expiration value greater than zero extends the life cycle of the subscription and results in a state-full approach, as the SIP server has to also keep transaction state for the subscriber. Moreover, the SIP server enforces a state-full approach for the processing of SIP PUBLISH requests as well.

2.1.2 Distributed and decentralized notification services

The deployment of event-notification services in the Internet today follows a centralized approach, where the communication of the events traverse a central notification server (i.e. SIP presence server). This approach exhibits certain limitations, mainly in terms of scaling and single point of failure. On the other hand, implementing a distributed and decentralized event notification service is a challenging task and has been thoroughly studied in the literature. The majority of the proposals [24, 25, 26, 27, 28, 29] addressing the issue propose a message delivery architecture that utilize some kind of peer-to-peer (P2P) technology.

In this thesis project we focus on topic-based publish-subscribe systems cooperating with a P2P routing infrastructure. Content-based equivalents are not considered in this work, since they incorporate an implementation and operational overhead when compared to topic-based systems. In addition, a smart publish subscribe strategy can also provision for synchronous operations while imposing minimum changes in the notification architecture. For example, SIP/SIMPLE provide a good proof of concept on how a publish-subscribe system tackles the issue of request-response information retrieval.

To the best of our knowledge, multicast technologies realize a topic-based and decentralized notification service. In a multicast system the subscribers join the same multicast group that the publishers use to publish their events. After an event is published into the group, multicast techniques defined in [30, 31, 32] are employed in order to efficiently communicate the event to the subscribers. However, even though IP multicast is not a new initiative and the vast majority of Internet devices actively support it (host multicast membership operations are integrated with the IP stack), it is not yet adopted for end-to-end use, as it was primarily intended. The decentralized Internet architecture, organized into independent autonomous systems, makes it difficult for any decisions that require holistic enforcement, like the support for multicast in an end-to-end fashion.

The unwillingness of the autonomous system authorities to implement IP multicast led the research interest towards Application Layer Multicast (ALM) equivalents. ALM utilizes a P2P network structure in order to efficiently deliver notification events to the subscribed peers. To the best of our knowledge the ALM solutions proposed so far, either construct their own P2P networks in a way that efficiently benefits the delivery of notification information [33, 34], or utilize existing P2P systems with proved functionality to deploy their operations [35, 36, 37]. The latter solutions have a cleaner yet more robust design since they separate their operations from the P2P system. By that way, one could potentially use different P2P network architectures for different deployment scenarios.

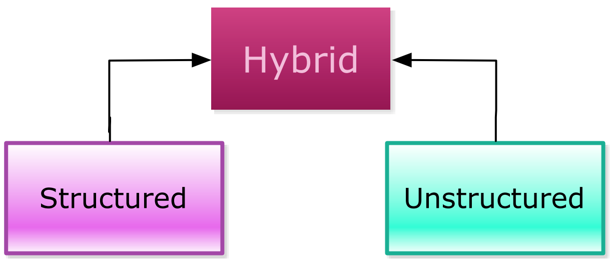


Figure 2.2: Classification of P2P networks based on their structure

A P2P network is a “virtual” network that is build on top of the physical one (i.e. IP network), by that way a P2P system builds an overlay structure among its peers. One way to classify P2P systems is trough their structural properties. According to Fig. 2.2 this classification ends up with structured, unstructured, and hybrid P2P systems. Structured P2P systems use a well-defined structure between their peers while unstructured use a near arbitrary peer formation strategy. On the other hand, the hybrid solutions (i.e. BitTorrent [38]) incorporate advantages from both the structured and the unstructured families of P2P networks, while trying to dispose their inefficiencies.

Structured P2P overlays provide hard delivery guarantees as a tradeoff for their increased organizational complexity and signaling overhead. However, for a structured P2P overlay network to meet its message delivery quality of service guarantees, a simple condition must hold; the P2P overlay must maintain its structure at all times. A P2P overlay maintain its structure when the routing table of the overlay peers reflect the structure of the overlay. When routing tables are not updated so as to reflect the overlay changes, the routing in the overlay fails and the overlay is said to be destabilized. Structured P2P overlays maintain the structure among their peers implementing a soft state approach, where each of the peers has to periodically check the availability of it's neighbors. It is self-intuitive that the tradeoff for message delivery guarantees is a complex P2P protocol structure along with an increased signaling overhead.

On the other hand unstructured P2P overlay networks form arbitrarily links between their peers, providing a best-effort message delivery service. Unstructured P2P architectures maintain a much lower operational complexity compared to their structured equivalents and are favorable for use with certain types of applications, like file sharing [39]. Even though unstructured P2P schemes perform really well for really popular information that usually resides within many peers in the overlay, they provide zero guarantees when they deal with the retrieval of isolated information.

As discussed in Section 1.3 the proposed architecture has to efficiently tackle the issues of resource location and context distribution. Resource location demands that the context information should be efficiently located and retrieved from the respective peer. In Section 1.1 the currently adopted distribution service perform resource location through a centralized point, the SPS. However in a distributed architecture the resources have to be distributed among the peers in a way to support load-balancing. Since unstructured P2P network overlays yield poor performance when locating isolated information, the requirement for resource location leads to the selection of a structured P2P overlay architecture. In addition, structured P2P systems are commonly used with ALM schemes to efficiently disseminate information (i.e. multimedia streaming). The latter feature of structured overlays is quite appealing, since it meets the requirements for efficient and robust context distribution. For these reasons our design choice is to implement a context distribution mechanism over a structured overlay network.

The rest of this chapter is organized as follows: in Section 2.2 we examine the major structured P2P systems proposed, where Section 2.3 reviews the research proposals in the domain of structured P2P systems and dynamic membersip. These two Sections set a number of requirements for the design of our architecture, regarding the desirable P2P overlay scheme. In Section 2.4 we review the state of the art multicast schemes and we state the requirements for the distribution service. Finally in Section 2.5 we elaborate on the research efforts of the IETF that enable SIP with distributed and decentralized features.

2.2 Structured P2P overlay networks

P2P systems tend to gain more and more attention since their attractive characteristics can lead the design and deployment of legacy client-server solutions towards distributed and decentralized equivalents. In a P2P system, the peers form self-organizing networks operating above the IP layer, often called P2P overlay networks or simply P2P overlays. These overlay networks provide a combination of appealing features such as: robust routing architecture, scalable location services, dynamic reconfiguration of the network as new peers join and existing peers leave, distributed and decentralized naming architecture and storage, authentication and security features, anonymity, scalability and fault tolerance.

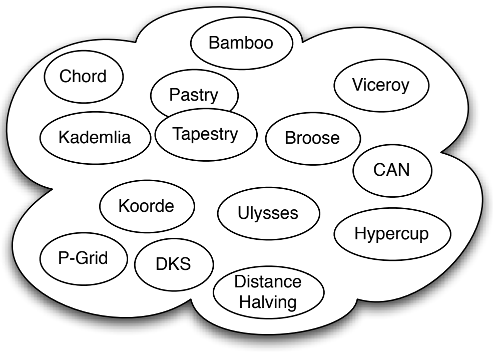


Figure 2.3: Some of the research proposals in the area of structured P2P overlay networks

As we have analyzed in the beginning of this Chapter and according to Fig.2.2, a P2P overlay network can be classified either as being structured, unstructured, or hybrid. An unstructured network needs almost no coordination since the links between the peers are arbitrarily formed, resulting in an overlay network geometry that does not follow any specific structural pattern. Consecutively, locating non popular information with flooding or with some heuristic algorithm yields poor performance. On the other hand, a structured P2P system increases the overlay's searching and routing efficiency with a tradeoff in an increased operational and implementation overhead. The bare bone functionality provided by a structured P2P overlay is that of a distributed hash table (DHT). That is given a key, the overlay algorithm maps that key onto a peer. Since each peer is responsible for a portion of the overlay's keys according to a specific assignment pattern, locating information within the overlay can be guaranteed with a bounded number of key lookup operations, thus, yielding a degree of quality of service. In the Internet today, P2P systems are of great interest where both academic and industrial bodies propose different structured overlay algorithms optimized towards different application domains (Fig.2.3). Even though these systems are realized differently, they inherit similar fundamental characteristics because of their DHT nature. These characteristics are reflected on the geometry of the overlay, the assignment of keys onto the overlay nodes, the lookup strategy, the mechanisms that maintain overlay's resiliency when churn occurs, replication and fault tolerance strategies, as well as the optimization mechanisms that form the peer interconnections based on their networking characteristics (i.e. RTT, bandwidth, etc.). In this Section we review and analyze the major structured overlay algorithms proposed but we first focus on these common characteristics.

DHT structure pattern:
Initially is the structure pattern of the overlay that reflects the overlay design. The structure pattern is the scheme of the overlay (overlay graph) used to create the connections among the peers. Some DHT proposals build a circular overlay graph that can be resembled as a communication ring, other mimic a butterfly network, where other model the overlay graph with a tree structure. The structure of the overlay affects many of the P2P system properties: such as the average routing path for the communication between two arbitrary peers,

the message delay, etc.

Mapping the keys onto nodes:
Besides the overlay graph each DHT system derives a different strategy for assigning the (key, value) pair of information onto peers. Briefly, each peer in an overlay is assigned an address space and is responsible for the storage of information with addresses belonging to that address space. The Internet equivalent to an overlay address space is the IP addressing scheme. In the P2P domain the address space is usually defined from the hash function used by the DHT. For example, a common hash function used by many DHT overlays is the SHA-1 [40] that generates hashed values of length $l = 160$ bits. With that length the address space consists of $\mathscr{K} = 2^{160}$ different addresses. Each address comes as the result of the hash function on the key of the (key, value) information pair and is used to route the (key, value) pair to the peer that holds the identifier space of that address. The identifier space assigned by the overlay to each of the peers is another important parameter that the designers of the overlay have to cater for. It is essential that each peer gets roughly the same portion of the initial identifier space in order to achieve good load balancing for the information stored at the peers. For that reason, each DHT tries to deliver identifier space chunks to each peer that on average hold $\mathscr{K}/N$ unique addresses. However, this is not a trivial task as the overlay is under a constant change (called *churn*) when new peers join and old peers leave at random time intervals.

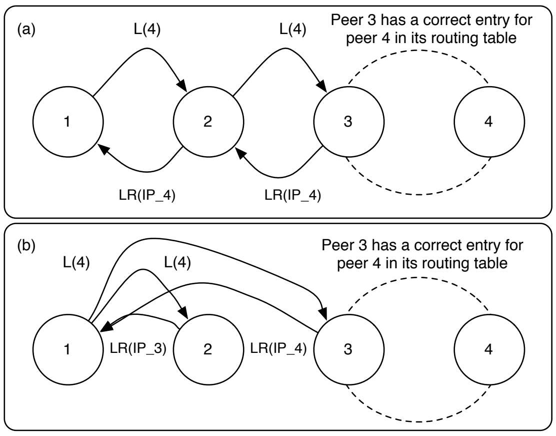

(a)
L(4)
L(4)
Peer 3 has a correct entry for peer 4 in its routing table
1
2
3
4
LR(IP_4)
LR(IP_4)
(b)
L(4)
L(4)
Peer 3 has a correct entry for peer 4 in its routing table
1
LR(IP_3)
2
LR(IP_4)
3
4

Figure 2.4: Recursive (a) vs iterative (b) lookup

Lookup functionality:
The third common property for a DHT system is the existence of a key lookup process. Through the lookup functionality a peer discovers which peer is responsible for what key. The lookup strategy is different for each DHT system realization since it is coupled with the overlay graph schema which reflects each peer's routing table structure. Moreover, key lookup is one of the system parameters that affects greatly performance quantities such as message delay, message delivery rates, etc. In theory, given any key its corresponding value can be located within just O(log$N$) hops, where N is the number of peers participating into the overlay. This property of the DHT-based P2P systems is very desirable, since it provides excellent scaling performance as the number of peers in the overlay increases. Every DHT-based overlay implements the lookup function using either the iterative or recursive variant. As seen in the Fig.2.4 (a), in the recursive lookup each peer propagates the lookup query to the peer it thinks that holds the answer regarding the requested key. When

the answer is found it is propagated backwards peer by peer to the peer that originated the query. On the contrary under the iterative lookup variant, which is depicted on Fig.2.4 (b), each peer replies to the peer that originates the lookup query its estimation about the peer that holds the answer for the lookup. Comparing both routing techniques we can say that the iterative strategy incorporates a number of advantages over the recursive one; it is easier to debug, consumes fewer resources on intermediate peers as no state information is installed for routing the response back to the originator of the routing query, and allows the querying peer to identify and route around misbehaving peers. However, in the presence of NATs iterative is intolerably expensive because a new connection must be established for each hop.

Churn resiliency:
Churn is the situation of overlay changes. As new peers joins and existing peers leave the overlay the structure of the overlay is altered, resulting in peers having obsolete routing tables that do not reflect the current overlay structure. Keeping each peer's routing table up to date is of major importance for every P2P system, as churn leads to destabilized overlays. A destabilized overlay degrades performance and certain procedures have to be invoked so as to become stabilized again. It worths mentioning that the quality of service message delivery guaranteed by the DHT-based overlays with generally $O(\log N)$ steps is achieved under a stabilized overlay. While the overlay is under churn, or even worse when there are malicious peers or peers showing byzantine behavior, the average required routing path between two arbitrary peers in the overlay becomes $>> O(\log N)$. Ideally, each research proposal in the domain of structured P2P overlay networks has to study and give worst-case guarantees when unpleasant situations like the one described occur in the overlay.

Replication and fault tolerance:
Finally, each DHT handles differently replication and fault tolerance issues. Replication is always desirable, especially when the overlay is intended to operate as a distributed storage pool, where access to distributed placed data should be offered at all times. Fault tolerance is a property of major importance for P2P system designers as it includes the techniques invoked when the overlay becomes completely destabilized and / or broken.

Proximity Neighbor Selection:
Proximity Neighbor Selection or PNS is the ability of a DHT-based P2P system to exploit IP network locality information when forming the overlay. In other words with PNS two peers that are overlay neighbors have also a small networking distance in the IP level. For example, let us assume that there exist a ring overlay with three peers (A, B, and C) that only establish communication in pairs. That is, A can only communicate with C via B and vice-versa. We also assume that peers A and C are under the same network domain and peer B is in a totally different network domain. In a DHT system where a PNS is not the case the peer positioning in the ring is made arbitrarily so a scenario like the one assumed can exist. The problem in such a scenario is self-explanatory, it is totally inefficient for two peers that are topologically close in the network layer to communicate through a third-party peer that exists in a network location far away from the peers that share good network locality properties.

Finally, a DHT-based P2P system provides the required abstraction layer to the applications running on top of it. By that way, and as seen in Fig.2.5, an application is exploiting a very simple Application Programming Interface (API) provided by the P2P core for manipulating information. Each application triggers get and set actions on (key, value) pairs of information and the DHT functionality running on each peer is responsible for locating and retrieving the requested information from the overlay.

It would be beyond the scope of this master thesis to evaluate every available structured P2P overlay that

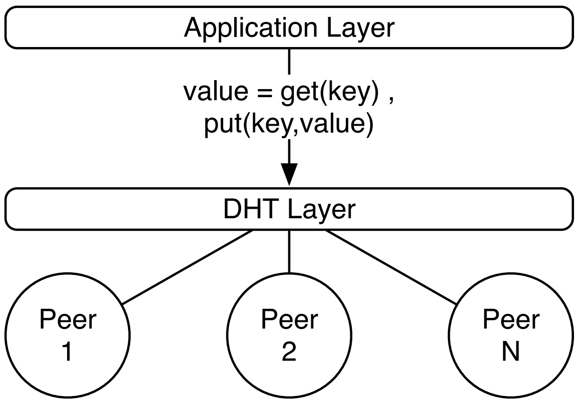


Figure 2.5: The fundamental functionality provided by a DHT

has been proposed from both the academic and corporate bodies. Instead, we focus on pioneering DHT algorithms that introduce some type of novelty. In the following subsections we evaluate the CAN [41], Chord [42], and Pastry [43] DHT overlay algorithms.

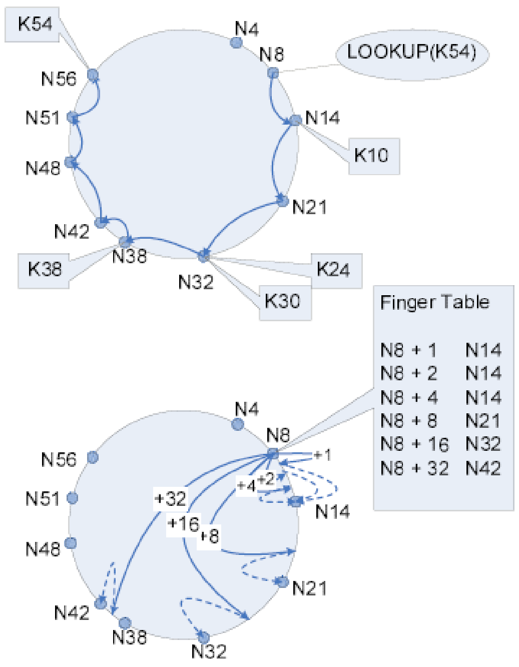
2.2.1 Chord

MIT's Chord [42] proposal builds a structured P2P network overlay utilizing DHT-like functionality. That is given a key, it maps that key onto a peer. Chord employs consistent hashing (using m-bit identifiers) in order to uniformly distribute the keys into the overlay network, resulting in load balancing. In Chord each peer is assigned a unique peer-id by hashing its IP address with a SHA-1 [40] hash function. The hashed value length (m) is selected in a way that the probability of two different keys have the same hash identifier is negligible. In the Chord implementation the identifier space holds $\mathscr{K} = 2^m$ addresses where $m = 160$ bits. The Chord system models the P2P overlay as a ring, where the identifiers are ordered into the ring modulo 2^m. That is, an arbitrary key identifier k is assigned to the first peer whose peer-id identifier is equal or larger than k in the identifier space. The peer which is assigned the identifier of the key k, is called the successor node of key k, successor(k). From the analysis presented in [42] for any set of N nodes and K key identifiers each node is responsible for at most $\frac{(1+\varepsilon)K}{N}$ keys and when the $(N+1)^{st}$ node joins or leaves the network only $O(\frac{K}{N})$ keys are being transferred to and from this node respectively. The upper bound for ε is proved to be $O(\log N)$.

The way Chord build the routing table of its peer in order to provide for scalable and efficient communication is a well addressed issue. Since Chord align the peers into the overlay using a circular overlay graph, each peer has to maintain its position in the ring by knowing each immediate successor and predecessor. However, using that token ring [44] approach Chord delivers information between two arbitrary peers into the overlay with $N/2$ hops on average. In order to increase performance and scalability as more peers join the Chord overlay, each peer has to maintain more routing information. It is a requirement though that the routing table should also scale in size as well. Chord's design guarantees an upper bound in the number of hops needed for the lookup of any key within the overlay with just $O(\log N)$ hops.

This can be easily understood by the following proof sketch. Let node n have a query for key k. The overlay node that is responsible for k is peer p. However, imagine a case where n have no access to k, and forwards the query to its closest neighbor towards p. Let that closest neighbor be f and also let f be in the i^{th} position of n's routing table. Thus the distance between n and f is 2^i. In case f needs to forward again the query to another intermediate or to p then this distance is at most 2^{i-1}, since the distance to the final destination can only decrease hop by hop. If $d_{old} = 2^i$, then $d_{new} = 2^{i-1} = \frac{2^i}{2} = \frac{d_{old}}{2}$, meaning that in the worst case the distance halves at each step. In the worst case within m steps (with m the exponent of the identifier base) the query will arrive at node p. Since the identifiers are random, after logN forwardings the probability is $\frac{2^m}{N}$ and the expected number of identifiers in this interval is 1 (with high probability). Thus, with high probability only O(logN) peers need to be contacted.

In Fig.2.6 an example of a Chord P2P overlay network is shown. In the upper Chord ring of the Figure an example lookup query originated by the peer with peer-id N8 is depicted. It is clear from this example that Chord implements a recursive lookup strategy. In the same Figure the finger table of peer with id N8 is depicted as well. Chord builds the finger table using a power exponent strategy, where the i-th entry points to the node $ID_{peer} + 2^{i-1}$. By that way the m-th entry point in a peer's finger table, points to an overlay peer 2^{m-1} hops away in the identifier space. All the arithmetic is modulo 2^m.



Finger Table

N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 + 16	N32
N8 + 32	N42

Figure 2.6: Example operation of the Chord algorithm

Chord handles node failures, implementing a soft state approach. That is every Chord peer runs periodically a stabilization routine. This routing has two legs. Primarily, stabilization has to ensure that the successor and predecessor pointer of a peer have to remain up-to-date as the peer membership in the overlay changes. By that way, each peer can accurately hold its position into the ring and at least forward lookups. The second leg of the stabilization routine checks the finger table to examine its consistency. Finger table maintenance runs with a lower rate compared to successor pointer maintenance, and this is mainly because the former is a demanding process in terms of network and computational overhead. When it comes to replication, Chord authors propose an r-redundancy scheme where every chunk of information (key, value) pair is stored in the respective peer and its r successors. By that way if any of the peers holding the same information fails, this information will still be available into the overlay. One of the major drawbacks of Chord, as presented in the original paper [42] is the lack for any PNS scheme. By that way two peers that are organized as logical neighbors in the Chord ring, successor-predecessor relationship, may be in totally disjoint locations regarding IP connectivity. This situation is totally undesirable both in large scale networks, as well as in small network domains, that are the case for our architectural design. Elaborate information about the entire structure of the Chord protocol can be found in the original paper [42].

2.2.2 Content Addressable Network

CAN [41] is another proposal for building scalable, fault-tolerant, and completely self-organizing P2P network overlays based on DHTs and is designed to provide Internet-wide scalability. Similar to Chord, CAN operates on its own address space, which is now called a $d-$dimensional coordinate system on a $d-$torus. This coordinate system is purely virtual and bears no relation to any physical coordination system. In CAN each of the peers is assigned its own portion of this $d-$dimensional coordinate space, which according to CAN's terminology is called a zone. Every (key, value) pair of information is mapped onto a point P in the coordinate space using a uniform hash-function. The corresponding (key, value) pair is then communicated to the peer that owns the zone that the point P lies

CAN also needs a scalable routing facility, for delivering the (key, value) in order to support for efficient key location. Every peer in the CAN P2P overlay network maintains in its routing table entries (IP address and respective zone) about its immediate adjacent peers in the coordinate space (i.e. the peers that their zones neighbor with the zone of that peer). In a $d-$dimensional coordinate space, two peers are neighbors if their coordinate spans overlap along $d-1$ dimensions and abut along one dimension. Moreover, the routing strategy in a CAN system follows the straight line path through the Cartesian space from the source to destination coordinates. According to [41] a CAN system achieves an average routing path length of $(\frac{d}{4})(n * \frac{1}{d})$ hops, which as the n increases it grows like O$(\frac{n}{d})$. In case the dimensions of a CAN system are selected according to $d = \frac{\log 2n}{2}$, then the average hops achieved by CAN are similar, O$(\log n)$, to the one of Chord [42].

In Fig.2.7, an example of a scaled-down CAN system using only a 2-dimensional coordinate space is depicted. The peer holding the zone $1c$, depicted in cyan, is the source peer routing a packet towards the destination peer. On the other hand, the destination peer holds the zone of the block $4a$, depicted in orange. The example route path of the packet is depicted with arrows and holds the route $2c, 2b, 3b, 3a, 4a$.

When a new peer wants to enter a CAN overlay it has first to bootstrap through some out-of-band methods a peer that already exists into the overlay. Technically the bootstrap peer may or may not be part of the overlay, as soon as it can establish connectivity between the joining peer and a third peer that already operates into the overlay. After the joining peer establishes communication with an overlay peer it creates a random point P and issues a JOIN request destined to that point that is injected into the overlay. Trough the CAN routing facility the JOIN message reaches the peer that holds the identifier space the point P belongs to. The peer that receives the JOIN message splits its identifier space (zone) in two halves and assign the one half to the

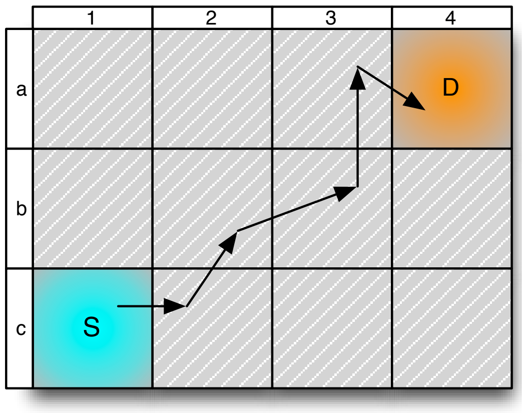


Figure 2.7: Scaled down version of a CAN system using a 2-dimensional coordinate space

peer that originated the JOIN request. By that way the joining peer obtains its zone as well as the neighbors of its newly acquired zone. After the joining peer contacts is neighbors is fully operational.

CAN exactly like Chord follows a soft-state approach for maintaining the integrity in the peers routing tables. In CAN each of the peers send periodic REFRESH messages among their neighbors for tracking down overlay changes, if a change occurs in the overlay and a peer fails or gracefully leaves the occupied zone have to be taken over by the remaining nodes. This procedure is thoroughly explained in [41] through the usage of a takeover mechanism.

The designers of the CAN system catered for two important aspects of structured P2P overlay networks, locality and replication. CAN implements a PNS mechanism with the introduction of multiple peers per zone. Based on the IP RTT metric a peer classifies each of the peers responsible for a single zone based on RTT delays. It follows that message forwarding between adjacent zones is performed using peers that score the best RTT metrics between these zones. Regarding replication of (key, value) pairs CAN proposes the $r-$reality mechanism. An $r-$reality consists of a CAN system having r $d-$dimensional systems, applying different hashing function per reality. By that way in a CAN with $r-$realities, a single peer is assigned r coordinate zones, one on every reality, holding r independent neighbor sets. The r parameter can also be seen as the replication factor, since a single (key, value) pair is stored r times on r different peers, thus achieving an $r-$replication scheme.

2.2.3 Pastry

Pastry [43] provides DHT-like functionality for creating a structured P2P overlay network and it appears great similarities with Chord [42]. In Pastry every peer is assigned an address belonging to a binary identifier space. Pastry uses SHA-1 [40] in order to map a key to a routable address. Even though Pastry, like Chord, places the peers in a circular manner, the identifier space used in Pastry does not follow a cyclic graph, rather it can be resembled as a tree.

In an overlay network consisting of N peers, Pastry can route a message to any destination peer into the overlay with just $\lceil \log_{2^b} N \rceil$ steps, under normal operation (b is a configuration parameter with typical value 4). The node ids and keys are considered as a sequence of digits with base 2^b. Pastry forwards messages to the peer whose node id is numerically closest to the given key. A peer normally forwards the message to a peer whose node id shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current peer node id. Each entry in the routing table contains the IP address of peers whose node ids have the appropriate prefix, and it is chosen according to close proximity metric. The proximity metric can provide a quantitative measure of locality, and it is based on the number of IP routing hops or geographic distance. The routing scheme found in Pastry, that performs a closer $b-$bits address matching at each hop, is primarily introduced in in the Plaxton et al. distributed search technique [45], and also resembles the longest prefix routing scheme found in the IP/CIDR architecture.

Each Pastry peer maintains a routing table, a neighboring set and a leaf set. The routing table consists of $\lceil \log_{2^b} N \rceil$ rows and $2^b - 1$ columns. The $2^b - 1$ entries at row n of the routing table each refer to a peer whose node id shares the present peer's node id in the first n digits, but whose $(n+1)^{th}$ digit takes on of the $2^b - 1$ possible values other that the $(n+1)^{th}$ digit in the present peer's node-id. Each routing table entry maps a node-id onto an IP address. Pastry can guarantee that when a new node joins or an existing node leaves the overlay, all routing tables can be updated (stabilization procedure) with the exchange of just $\log_{2^b} N$ messages. The leaf set $|L|$ is composed from the set of nodes with the $\frac{|L|}{2}$ numerically closest larger node-ids, and the $\frac{|L|}{2}$ nodes with the numerically smaller node-ids, relative to the present's peer node-id. The neighboring set $|M|$ contains node-ids and IP addresses for the $|M|$ peers that are closest (in terms of network locality (i.e. RTT)) with the current peer. In [43] it is shown that when peer failures occur into the overlay, Pastry can guarantee best-effort message delivery unless $\lceil \frac{|L|}{2} \rceil$ peers (with adjacent node-ids) of the leaf-set fail simultaneously. Typical values regarding the sizes $|L|$, $|M|$ of the leaf and neighboring sets respectively are $2^b, 2^{b+1}$.

Peer addition and departure (either voluntarily or due to failure) is an issue thoroughly described in [43]. Briefly when a new peer joins the Pastry network and generates its node-id (i.e. X), it obtains from the bootstrap peer a peer set with good network locality properties relative to X. Then, X asks from one of these adjacent peers to route a message with X as the destination peer. This message is routed to the existing peer with node-id Z that is numerically closest to X. X then obtain the leaf set from Z, and by that way it initialize its position into the overlay. In order to facilitate peer failures, each Pastry peer implements a soft-state approach with periodic message exchange for every peer entry registered in their leaf-set. By that way non responding peers are considered dead, and a stabilization procedure takes place in order to keep the routing tables of the remaining peers up to date.

In Fig.2.8 the route of an example message is depicted. In each step each peer forwards the message to a peer whose node id shares with the key a prefix that is at least one digit (or b bits) longer than the prefix that the key shares with the current peer node id. If no such key can be found the peer then forwards the message to a peer that shares the same prefix with the key as the current peer.

2.2.4 Kademlia

Kademlia [46] is another P2P overlay architecture that implements DHT functionality. The identifier space in Kademlia is partitioned as a tree and the XOR metric is used for the population of the routing tables. According to the XOR metric and for two keys (a, b) their respective XOR distance is $d(a,b) = a \oplus b \geq 0$ for

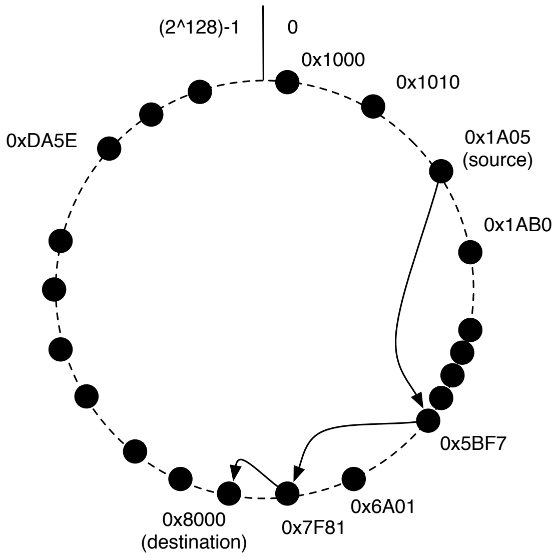


Figure 2.8: Operation of the Pastry DHT algorithm

$a \neq b$, thus the XOR distance represents the total number of bits that two keys differ. Kademlia constructs each peer's routing table as a collection of bucket lists. Every peer maintains as many bucket lists as the exponent of the identifier space (i.e. 160) and the routing strategy follows the longest prefix matching, as in Pastry [43]. The population of the bucket lists follows the XOR metric so that the i^{th} bucket list holds the identifiers that share the first 2^{i-1} bits but differ in the 2^{ith} bit with the identifier of the peer maintaining the bucket lists. Each of the bucket lists is able to hold k routing entries at full capacity, where k is a system-wide parameter. The bucket lists are constantly populated as packets originated from previously unknown peers flow through the overlay. However, Kademlia implements a "first in last out" bucket list update strategy, as it enables the overlay for efficiently handling DoS attacks targeted to routing table corruption. The key lookups in the overlay are implemented iteratively and concurrently, according to the concurrency parameter α. Each Kademlia peer issues α concurrent lookups per key in order to speed up the procedure with a tradeoff in the required signaling overhead. For a Kademlia peer to join the overlay the knowledge for a peer that already participates into the overlay is required (bootstrap peer). The joining peer initiates a lookup through the bootstrap peer using its own identifier as a key. The bootstrap peer responds with a list of the closely k-entries to the requested key. Finally, the joining peer perform updates on these k peers in order to populate its bucket lists further. Finally, it worths mentioning that Kademlia was developed after the initial DHT approaches (i.e., CAN, Chord) exposed the limitations in the design of DHT systems (i.e, security, lookup performance, etc.) and provisioned for these anomalies, thus yielding a solution intended to be used towards a real DHT deployment.

2.2.5 Discussion

In this Section we gave the common principles that govern a structured P2P overlay network and we have also reviewed the DHT algorithms that introduce some type of novelty. Table 2.1 summarizes the characteristics of the DHT algorithms under review. The advantages of each of the reviewed algorithms can be summarized as follows:

The Chord algorithm handles very efficiently the stabilization procedure by separately stabilizing the successor and the finger tables. Successor tables guarantee overlay structure and finger tables enhanced ($O(logN)$) key lookup. However, a small stabilization interval in the successor table maintains overlay integrity, combined with an increased stabilization interval on the finger table to accommodate the churn rate, results into an efficient (in terms of signaling cost) strategy.

On the other hand, CAN proposes an efficient replication scheme, that could be utilized by our architecture when it comes to resource location. By that way, a single information about context metadata could be replicated to r-nodes, in order to provide resilience and fault tolerance.

Pastry implements a PNS strategy that is very desirable in the design of our architecture. It is already mentioned in Section 1.2 that an efficient context distribution scheme should place the peers in the P2P overlay according to their network distance. In the proof-of-concept scenario of the same Section a maintenance worker heading to the incidents scene, should be enabled fast and uninterrupted context information flow, compared with an entity that is far away from the emergency.

Finally, Kademlia introduces a loosely structured P2P overlay that in the same time supports a robust routing infrastructure. In addition, Kademlia is one of the few overlay technologies, that is deployed and massively used.

2.3 P2P systems and dynamic membership

In the previous Section we analyze the common properties for different DHT systems and we review the major proposals in the area. However the evaluation of these systems as presented in their research papers usually assumes an ideal operating environment, where the overlay remains stabilized as no new peers arrive or existing peers depart. Studies [47] conducted on already deployed P2P systems showed that the effect of churn or alternatively known as dynamic membership is one of the biggest challenges for the designers.

The proposed context distribution architecture should operate even when the terminal availability is not guaranteed 100% of the time. In other words, when the terminals that build the distribution overlay experience intermittent connections (as known as churn) the overlay should be able to recover and continue provide service to the rest of the peers. In this Section we elaborate on the configurable parameters of DHT overlays that - when appropriately tuned - aid in defeating churn. These parameters define the size of the routing table, the stabilization interval, and the stabilization strategy. Moreover, when appropriately configured they can drastically vary the overall performance of the overlay.

Routing table size:
The routing table size is often directly controlled by the value of the identifier base and has an impact on the lookup performance. A large identifier base results in a small routing table, where a small identifier base result in a large routing table as their relation is given, in the general case, according to the formula: $O(log_b N)$, where b is a base 2 parameter. That is, the integrator of an overlay network can fix the identifier base to reflect the target capacity of the overlay. In a small overlay network, a coarse grain routing table granularity is more desirable than fine grain granularity, and vice-versa when the case is a large overlay network. The size of the routing table is an important parameter in the architecture of a P2P system, and

the designer must cater for an optimal value. An arbitrarily large routing table is able to store sufficient routing table information. However, this may result in redundant key entries pointing to the same physical peers. As the stabilization processes operate on the routing table, redundant routing table entries consume extra computational resources. On the other hand an arbitrary small routing table might not include sufficient routing information and thus deteriorate the key lookup functionality of the overlay. It is essential though for a DHT system to be able to adjust its routing tables for scenarios of small and large networks.

Stabilization interval:
The stabilization interval mainly affects the performance and the efficiency of the overlay, both in signaling as well as in computational overhead. The stabilization interval defines the rate λ_{stab} in which the stabilization routines are executed. The stabilization routines are of great importance for every overlay since they maintain the routing tables updated. However, in an overlay network with static membership where changes in the structure of the overlay happen rarely, keeping a small stabilization interval wastes network resources in terms of signaling overhead. On the contrary. in an overlay of dynamic membership and of high churn a high stabilization interval may result in a total destabilized overlay behavior and eventually to a total communication breakdown. It would be very desirable, if the P2P overlay of the proposed context distribution architecture could adjust it's stabilization interval according to the current churn load.

Stabilization strategy: A third factor with an impact on the overlay's performance is the stabilization strategy. An overlay can perform stabilization either proactively or reactively. The proactive strategy runs the stabilization routines every stabilization interval time units and thus implement a soft-state approach. In the proactive strategy every peer periodically checks the integrity of its routing table to maintain the structure of the overlay. In case a routing entry is outdated the stabilization routines on that peer are triggered. One of the main flaws in the soft-state approach is it's inefficiency mainly in terms of network and computational resources, which appear to be very valuable when the peer is powered by a battery source. On the other hand the reactive strategy implements the hard-state approach and triggers the stabilization routines only when a peer needs to use the overlay; usually prior to a key lookup. Reactive strategy enforces a resource efficient policy with a tradeoff in an increased message delivery delay. Analytical studies that compared the two strategies [48] pointed out that the reactive strategy is better than the proactive one when the churn load is low, but not when churn load becomes high. Thus, a state of the art DHT P2P system should implement a hybrid approach switching between the two strategies based on the churn load.

In the study of Li et al [49] different DHT systems are evaluated in a simulated environment under the presence of dynamic membership. For every of the DHTs a set of configurable parameters is defined and the value ranges for these parameters are given. Among other, these parameters affect the identifier base, and in turn the routing table size, as well as the stabilization interval, thus the computational and signaling overhead. Numerous simulation runs are set for each of the DHTs and the results are evaluated in terms of cost and performance. The quantities used to express cost and performance are the number of bytes sent per node, and the lookup latency on key requests respectively. Although, it is possible to derive other quantities for measuring cost and performance of the structured P2P overlay, the selected quantities provide a sufficient yet coarse grain evaluation. The DHTs used in the paper are the Tapestry [50], Chord [42], Kelips [51], and Kademlia [46], where Kelips follow the loosely structure nature of Kademlia offering satisfactory lookup performance, through continuous background communication between the peers (soft-state approach), in order to maintain the index structure of the overlay with high quality. Tapestry DHT can be considered as a Pastry variant [43] (i.e., the routing strategy employed in Tapestry differs than the one of Pastry - however both algorithms build a similar overlay graph). For Tapestry it is found that a lower identifier space base yields better bandwidth utilization as every peer maintains a smaller routing table and thus needs to contact fewer peers during stabilization. In particular, for bases 2 and 4 the nodes maintain the same amount of state however the latter base decrease the average hop count thus boosting the lookup queries. Regarding lookup latencies compared to different identifier space bases it is discovered that every base value is able to achieve the same lookup performance. Even though a higher base decrease the average hop count to any key in the overlay, the PNS mechanism of Tapestry equalize the lookup latency for various bases. It is outlined that *"In a protocol with proximity routing (PNS), the base can be configured as a small value in order to save bandwidth costs due to stabilization"*. On the effect of the stabilization interval on the performance of Tapestry, it is concluded that a more frequent stabilization is of priority since the cost in bandwidth is marginal when compared to the savings in lookup latency.

The simulations run on the Chord DHT unveiled that the finger stabilization interval affects performance without affecting the lookup success rate. However, a simple best case value for the stabilization interval does not exist, since there is always a tradeoff between lookup latency and signaling overhead with a varying stabilization interval. The lack of a PNS strategy in Chord reflects on the results for cost versus performance for different identifier space bases. Thus, larger bases result in lower latencies with an increased signaling overhead compared to the performance of lower bases. A rule of thumb for Chord, as mentioned by other research proposals [48], is to use a small base for high churn and a large base for low churn environments. For Kelips and Kademlia the study of Li focuses on the tradeoff between cost and performance having different stabilization interval values. In the conclusion of its study it is outlined that the stabilization interval should be aligned with the expected session time of the churning peers.

An alternative approach to the research proposals that are based on simulations is the recent study Krishnamurhty et al. presented in [52]. In this paper an analytical study of the Chord DHT in the presence of dynamic membership is given. This study acts complementary to the work presented in [48] and derives a set of formulas targeted for the configuration of DHT overlays. However, since the study is targeted to the Chord protocol it is doubtable how a system designer can tune a DHT system other than Chord, according to the given formulas.

In the study of Rhea et al. [53] the Bamboo DHT is presented. Bamboo can be considered as a Pastry clone, since it uses Pastry's overlay graph and routing algorithm. However, Bamboo is optimized in various aspects for efficiently handling dynamic membership. Bamboo is evaluated against a vanilla version of Pastry and MIT's Chord in a network emulator with varying peer session times ranging from 1.4 minutes up to 3 hours. The three most important factors taken into consideration for the Bamboo design is the:

- *Reactive versus proactive stabilization strategy*
- *PNS selection*
- *Timeout calculation strategy*

Regarding stabilization strategies Bamboo draws similar conclusions with the most recent studies based on analytical models. Reactive strategy is efficient for low churn, while proactive is suitable for deployments characterized with high churn loads. In addition, different PNS strategies are evaluated against cost and performance measurements similar to the one used in the study of Li [49]. The paper concludes that the different PNS strategies considered in the evaluation show similar behavior for both cost and performance regardless of their implementation and operational complexity. Thus, it is considered that a PNS strategy must be adopted since it drastically reduces lookup latency for a marginal tradeoff in bandwidth utilization, compared to a DHT that does not utilize PNS at all. Finally, the timeout strategy in terms of lookup latency was evaluated against a statically fixed timeout (5sec), a TCP-style timeout [54], and a timeout based on virtual coordinates. With the latter option a peer can dynamically calculate the lookup timeout of the queries destined to every other peer in the overlay. This functionality is achieved through a custom cost function that generates timeouts based on the virtual coordinates of the source and destination peers. It is essential that the virtual coordinates between the peers reflect their respective network delays, thus virtual coordinates enables for locality-aware timeout values. The yielded results are expressed in terms of performance and churn load and show that both TCP-style and virtual coordinated timeouts outperform statically fixed timeouts in the low and medium churn regions. Under high volume of churn the virtual timeouts perform better that their TCP-style counterparts, even though the latter perform better in the region of medium churn. Summarizing the work of Rhea et al., the findings pintpoint that a PNS mechanism as well as an efficient stabilization and timeout calculation strategy deeply affect the performance of an overlay.

A similar research effort to the one presented in the study of Rhea et al., derived MSPastry [55]. MSPastry is a Pastry based DHT overlay developed by Microsoft research optimized to combat churn. MSPastry incorporates various optimizations that reduce the signaling exchange between the peers and increase the routing efficiency of the protocol when used by Internet hosts.

2.4 Information dissemination

Up to this point we have focused at the transport infrastructure of the context distribution service. The actual context distribution service leverages the DHT overlay in order to deliver context notifications from the context producers (i.e. a context sensor) to the context consumers (i.e., a context-aware application). This section elaborates on different application level multicast technologies that can be used to distribute of information over a P2P network, and provision for peer failure and recovery features.

2.4.1 Information dissemination with the aid of multicast

Multicast technology is able to support scalable and efficient event notification services, as it enables data transfer from a group of senders to a group of receivers. Multicast functionality is used by a wide range of applications that want support for this kind of communication (i.e video conference, multimedia streaming, or networked computer games). In the domain of context distribution, multicast can be utilized to communicate context information originated from one or multiple context producers to a set of consumers.

In a multicast system the participating nodes form a communication tree, where the root node is usually the publisher. The information flow starts from the root to its children and is propagated down the tree until it reaches the leaf nodes. In such a multicast system, a membership mechanism that manages the structure of the tree as nodes join and leave the multicast group is required. Multicast was first introduced as part of the Internet protocol suite, IP, and a number of membership protocols were proposed that operate either between Internet routers [32, 56, 57], or between router and host [58]. However, for various technical and administrative reasons IP multicast has not been widely deployed. Application level multicast (ALM) came as an alternative to IP multicast, installing multicast functionality only at the end hosts. By nature ALM is inferior to IP multicast since it cannot guarantee that multicasted information traverses the same physical link only once. On the other hand, an ALM solution is a major resource saver compared to multiple unicast flows and can be easily deployed as the required functionality resides only at end hosts.

ALM solutions achieve information dissemination using tree or mesh based distribution techniques. Tree-based ALM schemes inherit operational primitives from their IP multicast equivalents and operate in a P2P basis. ALM proposals that maintain a tree structure can be further divided in ALM protocols that operate in conjunction to a structured P2P overlay [35, 36, 37] and protocols that act completely independently [34, 33]. Their main difference is that the first category exploits the structure provided by an already operating P2P network, where the second implements the tree structure using a fully independent membership protocol. It is self-explanatory that the first category is utilized in a system where a P2P overlay is already deployed, where the latter is able to provide a multicast communication package that can be used regardless of an overlay presence. However, exactly as in structured P2P system, the main challenge in the design of a tree-based ALM scheme is to maintain sufficient delivery guarantees as nodes join and leave the multicast channel.

Mesh based multicast protocols (i.e. Coolstreaming [59]) do not require an efficiently structured overlay, and thus exploit the unstructured paradigm. In a mesh-based multicast scheme each of the peers receives data from an incoming peer set (incoming degree) and further forwards that data to an outgoing peer set (outgoing degree). A newly joined peer forms the incoming and outgoing peer neighborhoods with the aid of a bootstrap node. The bootstrap node maintains the outgoing degree of all participating peers and defines the incoming peer neighborhood of the joining peer according to some predefined contraints. A mesh based protocol require minimum management and maintenance overhead and thus it can handle node failures better than a tree-based multicast scheme. In case the bootstrap node fails, the mesh-based protocol automatically stale.

The proposed architecture utilizes a structured P2P overlay to meet the resource location objective. Regarding context dissemination the choice is to utilize a tree-based multicast scheme instead of a mesh based, because the availability of the latter is coupled with the availability of the bootstrap node. Tree-based multicast can be applied either as a standalone solution or as an extension in the functionality offered by a P2P overlay. In our case since the overlay is introduced to handle resource location, it would be ideal to extend the provided functionality with multicast support. This design choice yields a multi layer architecture, where the overlay layer installs the routing and storage functionality and the multicast layer handles the information dissemination. However, the selection of a tree based multicast mechanism that better fits our application domain is not a trivial task, since there is a plethora of different research proposals building a multicast mechanism on top of a DHT-based P2P system [35, 36, 60, 61]. In the rest of this Section we will review the major proposals of the area and we will mainly focus on Scribe [35] a mature ALM solution designed to operate on top of Pastry [43].

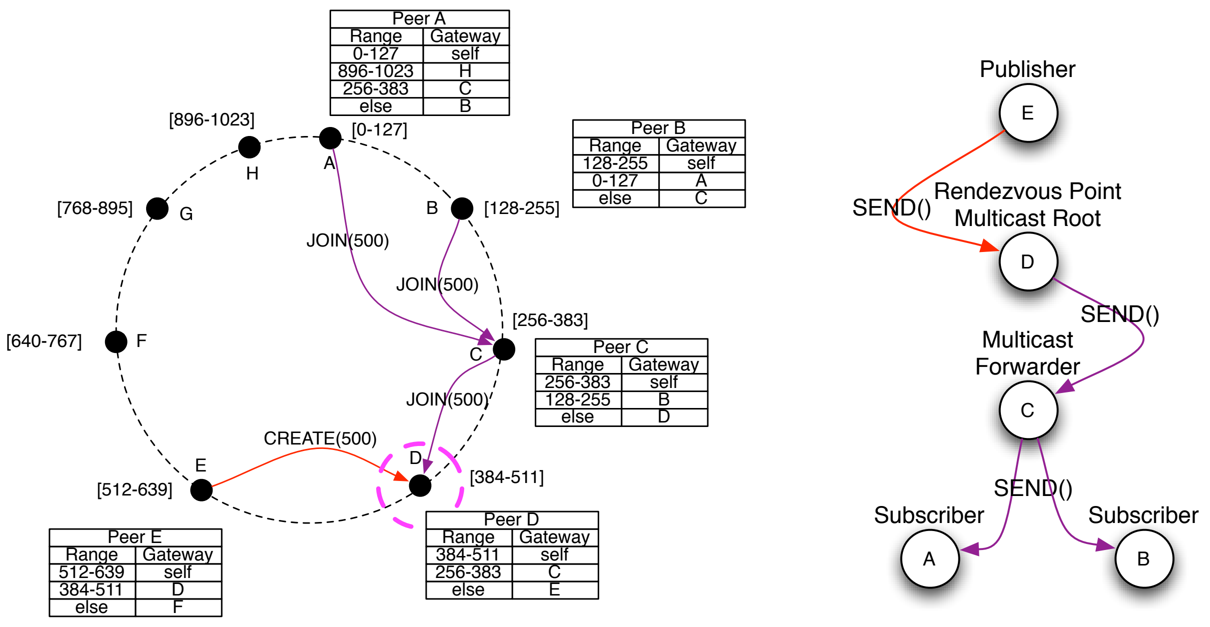
2.4.2 Scribe

Scribe [35] implements a topic based event notification service employing a tree-based approach and runs on top of the Pastry DHT. Scribe creates the trees using the Reverse Path Forwarding (RPF [62]) technique. Under RPF each multicast tree is formed by the union of the paths from the receivers to the root. Each multicast group in Scribe is identified by a groupID, which is the hash of the group's textual name concatenated with its creator's name and which results in a routable Pastry key. Fig.2.9a depicts an example overlay where peers A and B wants to join the multicast tree created by E. The overlay uses 10-bit identifiers and thus the identifier space consists of $\mathscr{K} = 2^{10} = 1024$ unique addresses. Moreover, the overlay consists of eight peers holding portions of the identifier space, where the portion held by each of the peers is further showed on the figure. In addition, the routing table maintained by each of the peers A, B, C, D, and E is depicted in order to facilitate the given example, where peer E is considered as the publisher and the initial creator of the multicast group. Initially peer E generates a name to act as the identifier of the multicast group; this name is further hashed using the overlay's functionality in order to form the group's unique routable identifier. In our example, the resulting 10-bit multicast address is selected to be number 500 (decimal).

For example, a query to key 500 ends at peer D, since that peer holds the portion of the identifier space with keys ranging from $[384-511]$. In other words, peer D is the responsible peer for the key 500. A Scribe peer joins a multicast group by simply sending a Scribe JOIN message towards the group id (gid). The gid is a routable key address and is the hash function outcome of an arbitrary identifier (i.e. when hashing the topic identifier "Alice's discussion group", using the overlay hash function, a routable key address (the gid) is produced (i.e., 500)). In the depicted example, peers A and B issue a Scribe JOIN message destined to $gid = 500$ in order to join the multicast group. The JOIN is routed within the overlay until it reaches the peer that is responsible for the message (peer D). Every intermediate peer that receives the JOIN creates a child entry in its multicast table holding the previous hop peer; by that way the multicast tree is formed. In the depicted example overlay peer C is the intermediate peer between peer D and the subscribers (A and B). Thus peer C installs an entry in it's multicast table stating that every received multicast information originated from peer D should be further forwarded to both peers A and B.

Following the Scribe terminology, the responsible peer for a gid (in our case peer D) is called the rendezvous point (RP) for that discussion group. In other ALM proposals the RP is often called the root of the multicast tree. It is self-explanatory that the peer creating the multicast group (peer E in our example) may be completely different from the RP peer (peer D). However, it is desired for some occasions to collocate the publisher with the multicast root functionality; Scribe comprise a set of techniques which guarantee that the initial creator of the group is also the RP of that group [35]. Fig.2.9b depicts the resulting multicast tree for the example given. It follows that as soon as the publisher, peer E, generates event notifications it communicates them to the multicast root (RP), using the overlay's routing infrastructure. From that point on the published events are further communicated down the tree, until they reach the leaf subscribers.

Scribe handles node failures using a soft state approach to maintain the tree structure. Each parent in the multicast tree periodically probes its children using a HEARTBEAT message (i.e. peer D polls peer C, and peer C polls peers A, B). A child suspects that its parent failed when it fails to receive HEARTBEAT messages. In case of failure the child issues a new JOIN message to the multicast root (RP) using the overlay routing infrastructure, and the tree is reconstructed. Sribe can also tolerate the failure of multicast tree roots (RPs). The state associated with each of the multicast groups stored at the RP is replicated across the k closest peers to the RP in the nodeID space. Scribe uses a replication factor of five (5) peers. If the RP fails, its immediate children detect the failure (the heartbeat timers expire) and join again using the overlay's functionality. The overlay routes the JOIN messages to the new root, which is the live node with the numerically closest nodeID to the groupID, which takes over the role of the RP. In order to remove stale children entries the parent anticipates that each of its children should send periodically refresh messages to sustain their subscription status.



Peer A

Range	Gateway
0-127	self
896-1023	H
256-383	C
else	B

Peer B

Range	Gateway
128-255	self
0-127	A
else	C

Peer C

Range	Gateway
256-383	self
128-255	B
else	D

Peer D

Range	Gateway
384-511	self
256-383	C
else	E

Peer E

Range	Gateway
512-639	self
384-511	D
else	F

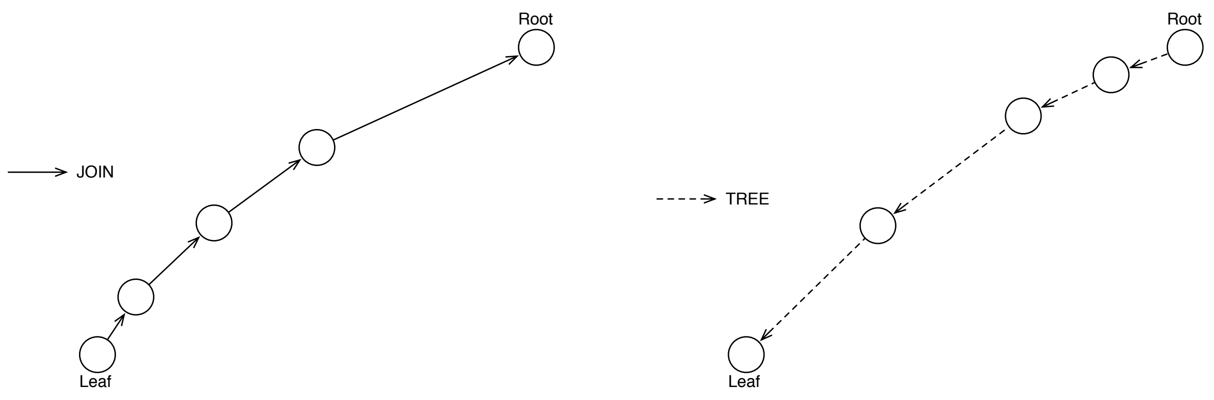
(a) **(b)**

Figure 2.9: An example overlay (a) where peers A and B subscribe to the multicast group created by E, resulting in a dissemination scheme that follows a tree structure (b)

Finally, Scribe introduces a mechanism to mitigate the creation of hot spots in the overlay; that is when a peer happens to forward excessive multicast information to a great number of children. Scribe employs a hot spot provision mechanism known as "bottleneck remover" [35]. In case a peer experiences high connectivity degree to a large amount of children it can force some of its children entries to re-assign their position in the multicast tree usually as grandchildren (i.e. children of it's children). Finally, Scribe exploits the locality provisioning feature provided by Pastry (PNS) to perform topologically aware dissemination. Thus, a parent-child relationship in the multicast tree experiences low delay communication cost, especially for the lower part of the tree links. However, this feature of Scribe which is a result of the RPF tree creation strategy is further explained in Subsection 2.4.4

2.4.3 Bayeux

Bayeux [36] is an ALM scheme that runs on top of Tapestry [50], providing tree-based multicast functionality. The multicast trees in Bayeux are created with the Forward Path Forwarding (FPF) technique. When a peer wants to join a multicast group it forms the groupID similarly to Scribe and routes a JOIN message towards the RP using Tapestry's routing infrastructure. The RP replies to the received JOIN with a TREE message destined to the joining peer. The TREE message is the equivalent to the JOIN, instead it is send down the tree from the root to the leaf, contrary to the JOIN that is send up the tree from the leafs to the root. As the TREE propagates to the destination peer the intermediate peers form the distribution tree. However, the difference in the tree building technique is not the only differentiation point between Bayeux and Scribe;



(a) (b)

Figure 2.10: The RPF (a) versus FPF (b) techniques when used for the creation of multicast trees

since the former protocol operates in a more centralized fashion than the latter. In Bayeux all the management traffic must go through the root of the tree, which also maintains a list of all the members participating in the multicast communication. Having all the membership information go through the root of the tree raises scalability concerns when the rate of subscriptions and publications increase. Bayeux proposes techniques that offload the multicast root, but the scalability of the architecture is only improved by a small constant factor [36]. Moreover, similar mechanisms to Scribe's "bottleneck remover" and multicast root replication are not available under Bayeux.

2.4.4 Scribe versus Bayeux

In order to directly compare the tree structuring techniques employed by Scribe and Bayeux we need to introduce a set of ALM performance evaluation metrics. *Link Stress:* measures the number of duplicate packets transferred by a selected physical link. For IP multicast link stress equals to one. *Relative Delay Penalty (RDP):* measures the path length of the overlay tree divided by the length of the direct path. For IP multicast RDP equals to one. Contrary to IP multicast where RPF and FPF create shortest path and low delay trees, their behavior in a routing strategy like the Plaxton [45] longest prefix one is quite different. The routing path achieved when RPF is employed consists of shorter and shorter edges moving from the root towards the tree leafs. On the other hand, the routing path achieved when FPF is employed consists of larger and larger links moving from the root towards the tree leafs. Fig.2.10 depicts the aforementioned effect of RPF and FPF techniques when used by Pastry and Tapestry. In Fig.2.10a the resulting RPF tree structure is depicted, whereas in Fig.2.10b the equivalent FPF tree structure is shown. As a result, messages traverse many long links near the leafs in Bayeux, while in Scribe, messages traverse few long links near the root. This is the reason that in their comparison [35, 60] Scribe scores better in link stress than Bayeux, even though both protocols score equal in RDP metrics.

2.4.5 Borg

A hybrid solution that incorporates both tree structuring techniques is proposed in [60]. Borg is an ALM architecture that operates in conjunction to Pastry and builds multicast trees using both RPF and FPF tree building techniques. Borg exploits the path assymetry provided by the Plaxton routing mechanism, inherited in Pastry and Tapestry, to construct the upper part of a dissemination tree using FPF and leverages the RPF technique for building the lower part of the multicast tree. The ultimate challenge that Borg faces is to find the optimal handover peer. From the depth of the handover peer and up to the root the tree is builded using FPF, whereas from the handover peer down to the leafs the tree is structured using RPF. Borg proposes a simply heuristic technique that discovers the handover peer in terms of overlay hops from the tree root. The performance evaluation of Borg [60] unveiled than the adopted hybrid architecture yields better performance metrics from both Scribe and Bayeux. Borg achieves better RDP scores than Scribe or Bayeux (on average half RDP unit less, than RPF and FPF schemes), where the achievable link stress is similar to Scribe's. However, Borg, as every hybrid architecture, inherits an increased implementation and operational complexity as the tradeoff for its enhanced performance, that given the allredy installed complexity of the underlying P2P system increases the whole complexity of the architecture.

2.4.6 Ubicast

Another proposal that implements a multicast tree to disseminate information is described in [61]. Ubicast builds a quality of service aware tree-based ALM scheme on top of Pastry [43]. Ubicast creates multicast groups identified by a Pastry routable groupID. Same as in Scribe and Bayeux a joining node issues a join message towards the RP associated with a groupID. However the RP in Ubicast simply redirects the join request to the group's creator (publisher), thus the multicast root is not the RP but the creator of the multicast group. Moreover, the root maintains the whole structure of the tree and controls every node operation. As soon as the root receives a joining request it derives the optimal place in the multicast tree for the joining peer according to some quality of service constraints; mainly in terms of bandwidth utilization [61]. However, the centralized design approach that enables Ubicast to efficiently disseminate multicast information raise scalability issues, especially when the target environment demands a fully decentralized architecture.

2.4.7 CAN multicast

An alternative proposal to Scribe, Bayeux, Borg, and Ubicast is specified in [37]. In that proposal an overlay-per-group multicast implementation is considered, where a secondary overlay operating on top of a CAN P2P system is created for every multicast group. The peer participates in the multicast group by simply joining the associated secondary overlay and flooding is used for disseminating information within the multicast-specific pseudo overlay. However, a comparison study between per-group-overlays and tree-based overlays concluded that the latter outperform the former in every measurement [63]. Furthermore, it is outlined that the only advantage of an overlay-per-group multicast scheme is that the multicast traffic is carried only by the members of the multicast group. In case this property is desirable for any administrative or other reason the overlay-per-group dissemination scheme can be considered, even though the measurements argue against it.

2.4.8 Discussion

The proposed thesis requires a scalable, robust, and fault tolerant context distribution architecture. Such requirements can be met with the selection of an appropriate ALM technology. In Section 2.2 a structured P2P overlay is proposed to support the resource location functionality. It would be desirable to extend the provided functionality of the P2P system with multicast capabilities. An ALM scheme that operates in combination with an underlying P2P overlay is the optimal choice. In the current Section we evaluate some of the major ALM protocols.

Among the reviewed protocols only Scribe has build in mechanisms that provide fault resiliency in case a peer of the multicast tree fails; multicast root included. Moreover, Scribe operates on a completely decentralized manner since the join/leave functionality can be handled lower the tree hierarchy and not only at the multicast root, which is the case for Borg. Moreover, the way Scribe builds the multicast tree is in favor of SIP, because both Scribe and SIP route messages using the RPF strategy. Thus, to the best of our beliefs, Scribe is able to provide a decentralized multicast solution as it offers a simple and robust design, while at the same time it is oriented towards real deployment scenarios (i.e., it provisions for data replication, RP failure scenarios, etc.). Moreover, the simple architecture of Scribe can smoothly incorporate (compared to a more complex multicast scheme) further extensions that may arise towards our goal for a distributed and decentralized notification framework compatible with the SIP/SIMPLE standards.

In Section 2.3 we reviewed the research proposals that address the topic of dynamic membership within structured P2P systems and concludes that churn is the biggest challenge for a system integrator. Since the proposed multicast functionality utilizes the key routing infrastructure of the underlying P2P system, it is anticipated to face service discontinuities in an environment of dynamic membership. The evaluation Chapter of this thesis further elaborateon the applicability of such a P2P multicast architecture with the presence of churn. The next Section presents the current research efforts of the IETF, that proposes a novel P2P architecture to substitute the legacy (centralized) SIP proxies / registrar functionality.

2.5 The IETF P2P-SIP initiative

Combining the functionality provided by SIP in a P2P system is not a new concept. Various research proposals [64, 65] exploit SIP for its appealing transport characteristics in order to exchange P2P related information. The SO/SIMPLE project [64] builds a structured P2P overlay network based on the Chord algorithm and uses ordinary SIP/SIMPLE functionality to transfer the overlay messages. The work proposed by Harjula et al. [65] extends SO/SIMPLE in the domain of mobile middleware. Similarly to SO/SIMPLE, Harjula [65] develops a middleware platform that organizes hosts in a Chord P2P overlay and uses SIP/SIMPLE messages to encapsulate the overlay's payload. Furthermore, the same work delivers context information among the Chord peers using ordinary SIP/SIMPLE operations. However, SIP/SIMPLE is used in a "centralized" configuration, where the requests are served directly by the context publishers. The work presented in this thesis further extends the concept of SIP/SIMPLE operations in a P2P network, since it introduces a distributed and decentralized notification architecture with the aid of multicast.

The IETF P2P-SIP working group extends the aforementioned research efforts towards a fully distributed and decentralized SIP architecture. More specifically, the working group tries to leverage the use of SIP in a P2P environment where the traditional proxy-registrar and message routing functions are replaced by some type of

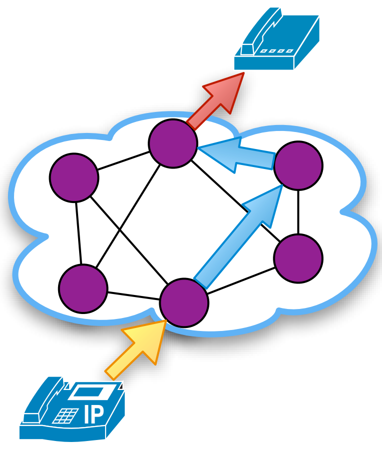


Figure 2.11: The vision for a P2P infrastructure offering SIP services

DHT functionality. This vision is depicted in Fig. 2.11, where standard SIP user agents communicate through a P2P network that substitutes the centralized SIP proxy/registrar entities, with distributed and decentralized equivalents.

One of the fundamental operations provided by SIP is the mapping of an Address of Record (AoR) for a user into one or more URIs [17]. The AoR is the name of the user completely decoupled from the host or hosts the user is using to communicate, while the URI indicates the host where the user can be contacted. Traditional or, the client/server SIP meets the resource location objective following a centralized approach, where a fixed hierarchy of SIP proxys/registrars are contacted to perform the mapping between AoRs and URIs.

P2PSIP needs to break the centralized architecture imposed by client/server SIP and distribute the functionality provided by this fixed hierarchical scheme among the peers of the overlay network. Thus, in P2PSIP the overlay peers offer storage and transport services to allow the distributed database function and distributed transport function to be implemented. However the mapping between AoRs to URIs is only the bare-bone functionality offered by traditional SIP. The vital components in a SIP network consist of third party services (i.e. STUN service, SIP to PSTN gateway service, SMS gateway service) that add value and usability in a SIP network. In P2PSIP the peers advertise their third party services in the overlay in order to become accessible to every other peer.

Fig.2.12 depicts a proof of concept P2PSIP overlay adopted by [4], where the various overlay peers provide coupling with traditional SIP services. According to Fig.2.12, Peer F provides a SIP to PSTN gateway service. Subsequently Peer F has to advertise the offered service in order to be accessible by the rest of the overlay peers. As the overlay provides DHT functionality, Peer F simply has to add the URI describing the offered service into the overlay's distributed database. A peer that desires access to the PSTN service simply performs a key lookup into the distributed database, using as a key the service URI. In the same Figure, Peers P and Q are shown behind a firewall / Network Address Translator (NAT), which is a usual case in a real overlay deployment; thus P2PSIP raises demands for efficient and practical firewall / NAT traversal mechanisms between the peers. P2PSIP's intention is to exploit the well-tested NAT penetration mechanisms used by traditional SIP [18, 20].

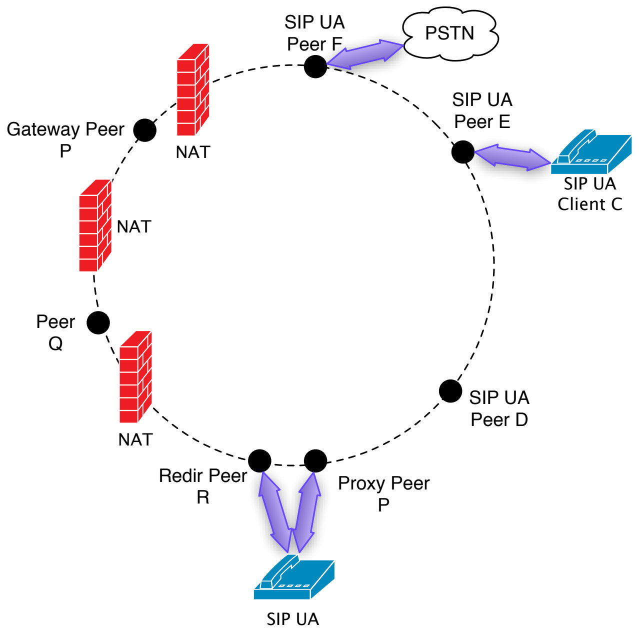


Figure 2.12: P2PSIP overlay reference model [4]

P2PSIP also supports devices with limited connectivity (i.e. behind a strict firewall) or with poor resources treating them as overlay *clients*. The role of clients in the overlay is still under much debate but the group has reached consensus that clients will be able to store and receive information from the overlay. As the P2PSIP working group is still in a premature state, a plethora of vital issues still remains open. Along with the role of clients every other P2PSIP functionality is still at the drawing board. The issues need to be addressed regarding the role of the clients, the protocol spoken between the peers, the protocol spoken between peers and clients, the coupling with traditional SIP, as well as NAT traversal requirements. The majority of these issues are addressed with various individual draft documents that provide proof of concept solutions. One of the most important parts of P2PSIP is the resource location and routing architecture (RELOAD) [5].

The RELOAD framework [5] provides the resource location and discovery capabilities to a P2SIP network. RELOAD is also the protocol spoken between the overlay peers, and defines a multi-tier, open, and pluggable peer protocol architecture, which is depicted in Fig. 2.13. According to that Figure the proposed architecture is comprised from different components, namely: the routing layer, storage, topology plugin, and forwarding layer. The routing layer is responsible for routing the messages through the overlay; it can be seen as the implementation of the peer's routing table since it directly serves the key lookup requests originating from the topology plugin as well as the store and fetch queries initiated from the storage component. The storage component is responsible for processing the messages related to data storage and retrieval actions; it is the realization of the distributed database and it talks directly to the topology component and the routing layer, in order to send and receive messages and manage data replication and migration. The topology plugin implements the specific overlay algorithm being used and currently incorporates a modified version of the Chord algorithm.

Moreover, the proposed overlay algorithm specifies a reactive stabilization strategy for the leaf set and periodic stabilization strategy for the finger table. By that way RELOAD achieves overlay connectivity even in cases where high churn occurs and message quality of service guarantees when low churn is the case. Given the fact that RELOAD is primarily targeting static environments (i.e. P2PSIP as a PBX replacement technology) with a relatively small number of hosts, the specified stabilization strategy fits the considered application domain. However, RELOAD is not adopting any PNS strategy for the arrangement of the peers in the overlay. Even though PNS does not boost overlay's performance in case the peers reside within the same access network, it would be efficient if the core overlay algorithm provisions that feature.

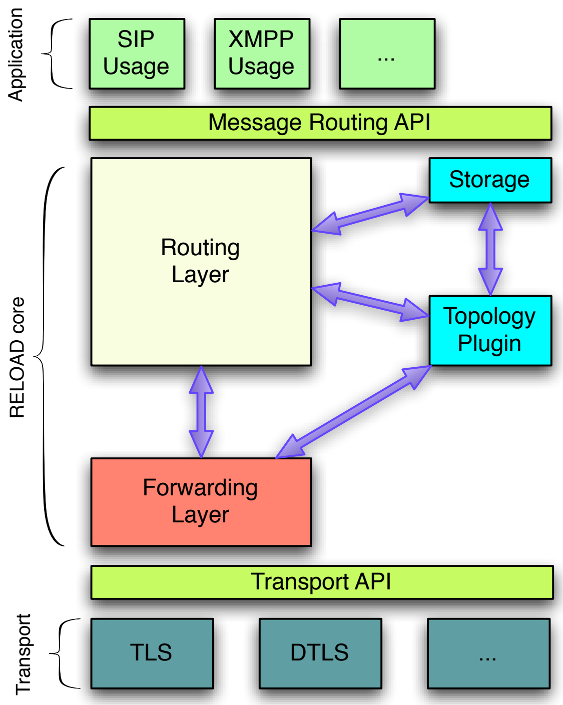


Figure 2.13: The RELOAD architecture [5]

In addition and according to Fig. 2.13 the functionality provided by RELOAD resides between the transport and the application layer. RELOAD is intended to act completely decoupled from SIP, thus yielding a decentralized database and transport package for general purpose use. The architecture adopted by RELOAD reflects the design decisions of the working group in terms of the P2P service scope. Initially the group considered SIP as the peer protocol, influenced by the legacy research projects described in [64, 65]. As the discussions in the group evolved, P2PSIP reconsidered the tight coupling between the peer communication protocol and SIP, eventually proposing the leverage of a custom, lightweight, and binary peer protocol. A binary peer protocol (i.e., DNS, SNMP) uses binary encoding in both the protocol headers and payload information. On the other hand human - readable protocols (i.e., SIP, HTTP) use text based headers that are human readable, thus easier to understand, deploy, and debug. Nevertheless, binary protocols produce less signaling due to smaller packet sizes, and are in general faster compared to human readable protocols. This is because the binary parsing operations are much faster than the string equality operations required while parsing a human readable protocol.

The concept of RELOAD exhibits great similarities with the distributed event notification architecture proposed in this thesis; since both schemes utilize a P2P system to enable for scalable and efficient resource location capabilities. Moreover, the RELOAD proposal, as being an IETF document, is designed to address issues (i.e., authentication, security, replication, NAT traversal, etc.) that the proof of concept academic papers, including this thesis project, usually keep aside for the sake of simplicity. Thus the RELOAD specification elaborates to a great extend, far more than the research proposals tackling the same issues. The P2P architecture proposed in this thesis to efficiently meet the objective of resource location could potentially utilize RELOAD as the P2P infrastructure, since the latter provide a general purpose distributed database and routing package oriented towards real deployments.

To sum up, this Chapter gave the background information on structured P2P overlay algorithms, and ALM protocols. Moreover, we reviewed the relevant research in the field of dynamic membership in P2P systems and we presented a guideline for the configuration of such systems towards churn resilience. Finally, the P2PSIP initiative that focuses on decentralized SIP proxy / registrar functionality was given. The following Chapter presents the proposed context distribution architecture in detail, where the simulated effect of churn in our architecture is presented in Chapter 4.

Structured overlay algorithms summary				
Properties	CAN	Chord	Pastry	Kademlia
Iterative/recursive lookups	iterative & recursive	iterative & recursive	iterative & recursive	recursive
Lookup performance	$O(dN^{1/d})$	$O(logN)$	$O(\log_b N)$	$O(\log_b N)+c$
Routing table state	$2d$	$logN$	$2b\log_b N$	$b\log_b N+b$
Maintenance overhead	$2d$	$\log^2 N$	$\log_b N$	$\log_b N+c$
Replication/fault tolerance	yes/yes	yes/yes	yes/yes	yes/yes
PNS strategy	yes	no	yes	no
Security	not considered			DoS resilient
Appealing feature	Redundant DHT replication scheme	Efficient stabilization procedure	Smart overlay structure with the aid of PNS	Loosely defined structure and robust key routing

Table 2.1: Structured overlay algorithms summary

Chapter 3
System architecture

3.1 Introduction

The SIP support for event notifications (RFC3265 [2]) and event state publications (RFC3903 [1]), follows a topic based system approach. Thus, a topic identifier is used to uniquely identify different notification groups. In case of SIP, the Request URI header of a SIP message serves as the topic identifier. Notice that in this Chapter the term notification server refers to the SIP entity that processes publication and subscription requests and communicates the publications to the subscribers as SIP notifications.

Given a fixed publication rate the notification overhead experienced by a SIP server is proportional to:

$$\text{Notification overhead: } \propto \textstyle\sum_{i=1}^{N} S_i$$

where N is the number of notification groups, and S_i is the number of subscribers of the i-th notification group.

A state of the art notification architecture that meets the objectives for scalability, redundancy, as well as for failure and recovery is a challenging task. We argue that decentralizing the event notification process within a P2PSIP overlay network yields a scalable architecture able to accommodate an arbitrary large number of subscribers and notification groups. With a P2PSIP notification architecture we can distribute the notification groups and the actual notification load (overhead caused by subscription and notification signaling) within the overlay peers.

In order to decentralize the notification groups the hash and key based routing functionality of the overlay is used. Hashing the topic identifier of a notification group (i.e. SIP Request URI) produces the gid which is a routable key address. The hash function of the overlay guarantees that in the long run in an overlay of N peers and K notification groups each of the peers gets responsible for $\lceil \frac{K}{N} \rceil$ notification groups.

Additionally, the actual notification load can be distributed with the aid of a tree-based application level multicast protocol. Such an improvement offloads the multicast root as the subscriptions can be handled lower the tree hierarchy. It also reduces the per link notification signaling as only one notification is needed to travel down the tree and received by the different subscribers. Regarding the notification delay, an upper bound can be considered given the fact that the length of a multicast tree is in the order of O(logN) hops, where N is the size of the overlay.

The fundamental components of our architecture are a structured P2P overlay network, an application level multicast protocol, and a modified SIP stack. The SIP stack is used to receive SIP requests from non overlay SIP user agents and invoke the P2P SIP functionality. However, the way these components are interconnected yields analogous realizations (variations) of the architecture. In the subsections that follow, two different variations of the architecture are presented according to different design objectives.

The sections that follow present two different design approaches of a P2PSIP event notification architecture. The first, initial, approach presents a system that should reuse different overlay algorithms and ALM protocols based on open Application Programming Interfaces (APIs). In that approach a "SIP/SIMPLE wrapper layer" is proposed that accesses the ALM and the overlay through open APIs and translates standard SIP/SIMPLE calls to overlay equivalents. As it turned out, the "wrapper" layer needed access to internal state maintained by the ALM protocol that could not be accessed through the standardized open calls. At that point any modification either to the ALM protocol (in order to expose that state) or to the open APIs violated the initial requirement for component reusability. For that reason, we revised the architecture into an alternative that merges the ALM protocol with the "wrapper" layer. The latter, refined, approach is presented in Section 3.3.

3.2 Variation I - A cross layer approach

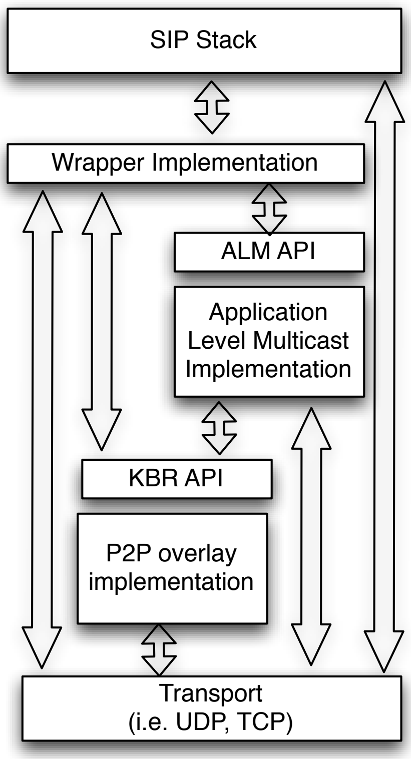

SIP Stack
Wrapper Implementation
ALM API
Application Level Multicast Implementation
KBR API
P2P overlay implementation
Transport (i.e. UDP, TCP)

Figure 3.1: Variation I - Full cross layer approach

The initial design follows a cross layer approach and is shown in Fig. 3.1. Our primary objective is to use the architectural components off-the-shelve. This produces an agile and extensible architecture. Extensibility is achieved, since the P2P overlay algorithm and ALM protocol are accessed through a collection of public and open Application Programming Interfaces (APIs) [68, 69] that allows the architecture to operate with different component realizations. Thus, different application level multicast and P2P overlay implementations could be leveraged as long as they comply with these APIs.

The architecture component that coordinates the dissemination procedure is a custom "wrapper" protocol that maintain communication interfaces with the transport layer (i.e. TCP, UDP), the P2P overlay, and the ALM protocol. The main functionality of this "wrapper" protocol is to match the behavior of SIP event notifications and distribute that process among the overlay peers. An upcoming issue is that the SIP notification model is primarily synchronous and optionally asynchronous according to the value of the SIP *Expires* header of the SIP SUBSCRIBE request. On the other hand an ALM protocol provide only asynchronous notifications; the user subscribes to the ALM group and as soon as a publication is made to the group, a notification reaches the group members.

Since this approach demands for modularity, access to the ALM implementation is made through the provided API calls and any custom modification to the ALM protocol implementation breaks the extensibility of the architecture. Consecutively, the ALM protocol is leveraged to serve the asynchronous part of SIP event notifications, where the mandatory synchronous part is served by the "wrapper" protocol. Moreover, it is of primary concern that the signaling path serving the synchronous notification requests matches the asynchronous path that is created and maintained by the ALM protocol. In order to assure that the "wrapper" and ALM protocols operate on the same path for every notification group, our custom protocol has to follow the path creation techniques of the ALM protocol.

Application level multicast protocols inherit path creation techniques already available in traditional (IP) multicast schemes. Therefore, the majority of the protocols build multicast trees using the reverse and forward path forwarding techniques. Our intention is to use the reverse path forwarding technique for building the path, not only because it better fits certain popular overlay algorithms [63], but because the SIP proxy model operates in the same fashion. When a SIP message is routed through multiple SIP proxies, each of the proxies appends its address in the *Via* header field of the request. The SIP response then follows the same proxy list in its reverse order and the requirement for equal path traversal between SIP requests and responses is met. Thus, the SIP routing strategy operates in an RPF fashion.

3.2.1 Inherited limitations

The appealing advantages of a full cross layer design approach are unbeatable. However, the strict decision to utilize the peer overlay and ALM protocol through open APIs inherits severe limitations to the architecture. This design approach mandates that the custom protocol as well as the ALM protocol create and maintain the same path to disseminate notification requests. As a result the architecture installs extra computational and network overhead to the overlay peers, as exactly the same path state needs to be maintained in two protocols running on the same node. In addition, the decision to adapt the reverse path forwarding technique by the wrapper "protocol" violate the extensibility objective as non every ALM protocol implementation can be plugged into the architecture. Since our initial objectives cannot be met and at the same time produce an efficient P2P SIP event notification architecture we consider an alternate realization of the architecture, that is not extensible, inherits a monolithic design, but its overall simplicity overcomes any initial concerns.

3.3 Variation II - A monolithic approach

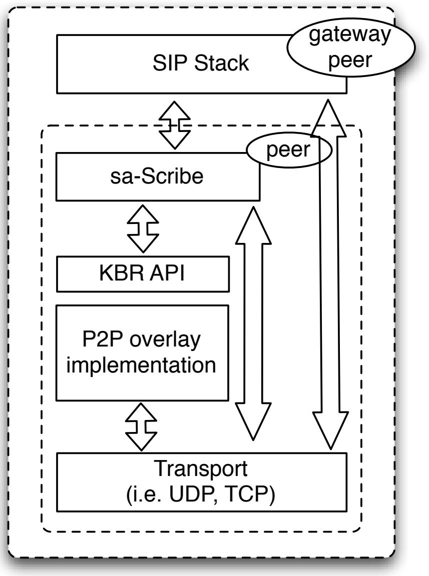


Figure 3.2: Variation II - A monolithic approach

The second, alternative, design is based on a monolithic architecture and targets a simple process that achieves SIP based event notifications within a P2P overlay network. The components of this realization and their connectivity is shown in Fig. 3.2. This design approach leverages a custom application level multicast protocol, that is based on Scribe [35], and extends it to support SIP like notifications behavior. Scribe is an application level multicast protocol that builds dissemination trees based on the reverse path forwarding technique, first implemented in IP multicast. Although similar realizations of tree based application level multicast protocols exist ([36, 27]), the advantage of Scribe is the ability to decentralize the entire notification process, while catering for fault tolerance through state replication; multicast root inclduded. Comparative studies [63] show that Scribe outperforms similar application level multicast implementations since the employed reverse path forwarding technique exploits the routing asymmetry of certain popular overlay algorithms (i.e Pastry, Chord) resulting in better a performance.

However, Scribe implements a multicast behavior: it builds multicast trees and asynchronously disseminates information down the tree on the reception of a multicast call. We would like to make Scribe SIP-aware, in order to implement decentralized and scalable notification server functionality. To achieve this we needed to change Scribe's behavior by following the SIP notification model. This results in a new protocol called SIP aware Scribe (sa-Scribe), that adds Scribe a new functionality to fetch the latest known notification state on a new subscription call (i.e. the synchronous behavior).

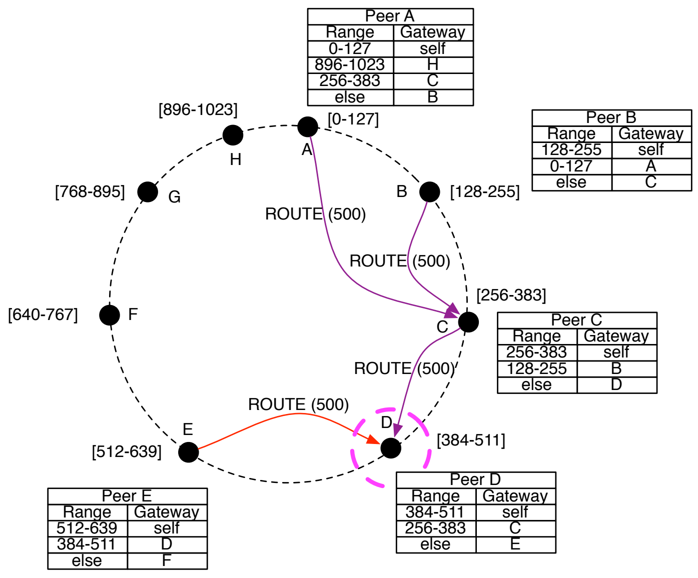
The need for synchronous response in Scribe increases the complexity of the protocol, and a simple acknowledgment mechanism similar to the one leveraged by SIP is needed to control the request and responses between sa-Scribe peers. Sa-Scribe needs to support three types of acknowledgements: the positive

(ACK), the negative (nACK), and the provisional (pACK) acknowledgment. Provisional acknowledgements are needed because a sender often routes a message through multiple intermediate overlay peers without a-priori knowledge of the multicast root. A pACK maps to the SIP response codes 1xx, an ACK to the SIP response codes 2xx, where a nACK covers the SIP response range [3xx, 6xx]. Along with the acknowledgments an ordered transmission scheme is considered (leveraging sequence numbers).

The sa-Scribe also changes the way publications in the overlay are performed. The create and multicast calls of Scribe get merged into a new publish call, which creates a new notification group (if none exists) for a particular gid, and also carries the message to be multicasted to the participants of the group. In order to comply with the SIP PUBLISH behavior, sa-Scribe's publish is enabled entity tag (etag) support. The use of etags enables multiple publishers that publish information under the same gid, to know if their publication is the latest or if another publisher generated a more recent. This functionality is very desirable especially in situations where multiple publishers need to cooperate their publication actions without having to subscribe for their own publications. The actions performed by sa-Scribe on the reception of publish and subscribe calls are given in the Subsection 3.3.2 and 3.3.3.

3.3.1 The overlay network

The internal formation of the overlay network used in our message sequence charts is shown in Fig. 3.3. It is assumed that the overlay utilizes 10-bit long addresses resulting in an identifier space of K = 2^{10} = 1024 unique addresses. The overlay consists of eight peers, each one holding equal portions of the original identifier space. Both the address space assigned to each of the peers as well as their respective routing tables are shown.



Peer A	
Range	Gateway
0-127	self
896-1023	H
256-383	C
else	B

Peer B	
Range	Gateway
128-255	self
0-127	A
else	C

Peer C	
Range	Gateway
256-383	self
128-255	B
else	D

Peer D	
Range	Gateway
384-511	self
256-383	C
else	E

Peer E	
Range	Gateway
512-639	self
384-511	D
else	F

Figure 3.3: Internal deployment and connectivity of the P2P SIP overlay utilized in the example

For example a route operation to the routing address 500 initiated from peers A and B would result at peer

D, through peer C. On the contrary the routing table of E contains the information that for the key 500 responsible peer is D (since D's address space spans over the key range [384, 511]).

3.3.2 Message sequence chart - Publication

As shown in Fig. 3.4, peer E receives a SIP PUBLISH request and responds with a SIP 100 Trying provisional response. Additionally, peer A invokes a local publish call to the sa-Scribe protocol, providing the gid, an expiration value, and the publication data. In case the publish call from Alice carried etag information the etag value should be included in the local publication call. The publish call reaches the multicast root for that gid (peer D) and a new state for that publication is created (if there is not any). Subsequently, sa-Scribe running on D acknowledges the reception of the publish call with a positive acknowledgment that carries the gid, the final expiration value as decided by D, and the etag information. Upon the reception of this response peer E composes a SIP 200 OK response, which is propagated back to Alice.

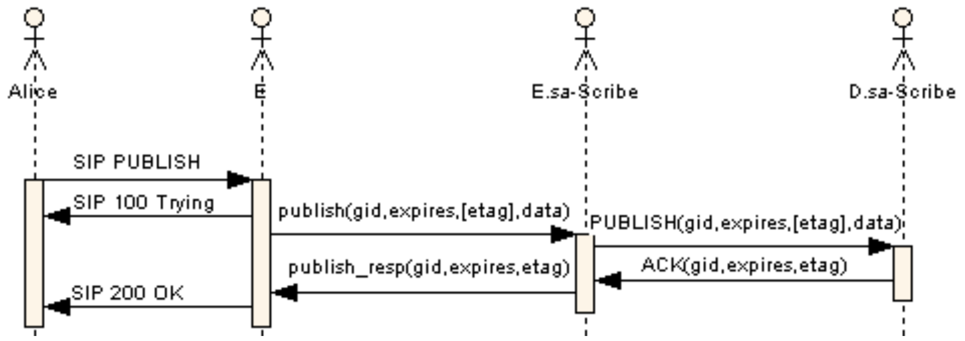

Alice
E
E.sa-Scribe
D.sa-Scribe
SIP PUBLISH
SIP 100 Trying
publish(gid,expires,[etag],data)
PUBLISH(gid,expires,[etag],data)
ACK(gid,expires,etag)
publish_resp(gid,expires,etag)
SIP 200 OK

Figure 3.4: Publishing information into the overlay

3.3.3 Message sequence chart - Subscription

The actions performed by sa-Scribe on the reception of a subscribe call are shown in Fig. 3.5. Following Fig. 3.5, the non overlay user agent Bob sends a SIP SUBSCRIBE request to the overlay with first point of contact peer A. The modified SIP stack of peer A issues a subscribe call to the sa-Scribe protocol running on the same node providing the gid and the proposed expiration value as arguments. Consecutively, the sa-Scribe issues a JOIN message and routes it through the overlay towards gid. The sa-Scribe protocol running on peer C receives the join and replies back to A with a provisional acknowledgment, while it further routes the JOIN request in the overlay towards D; which serves as the multicast root for this subscription.

In case D maintains event state for the gid of the JOIN request, an ACK and a subsequent NOTIFY carrying the latest event state are returned to C and eventually to A. On the contrary, if D maintains no state for that gid a NACK is returned first to C and then to A. On the reception of an up-call from sa-Scribe the SIP stack of A finds the associated SIP transaction state and communicates the final SIP response back to Bob. Any notification requests are communicated from D to Alice in the same fashion.

In order to show the decentralization features of the proposed architecture, let us consider the following case that extends the previous example. In case peer B wants to subscribe to gid 500, that SUBSCRIBE call will not reach the gid's RP (peer D). Instead, peer C that already maintains the latest state for that gid (through the subscription action previously originating by peer A) will reply directly to B, acknowledging the subscription request and further providing the latest known event state.

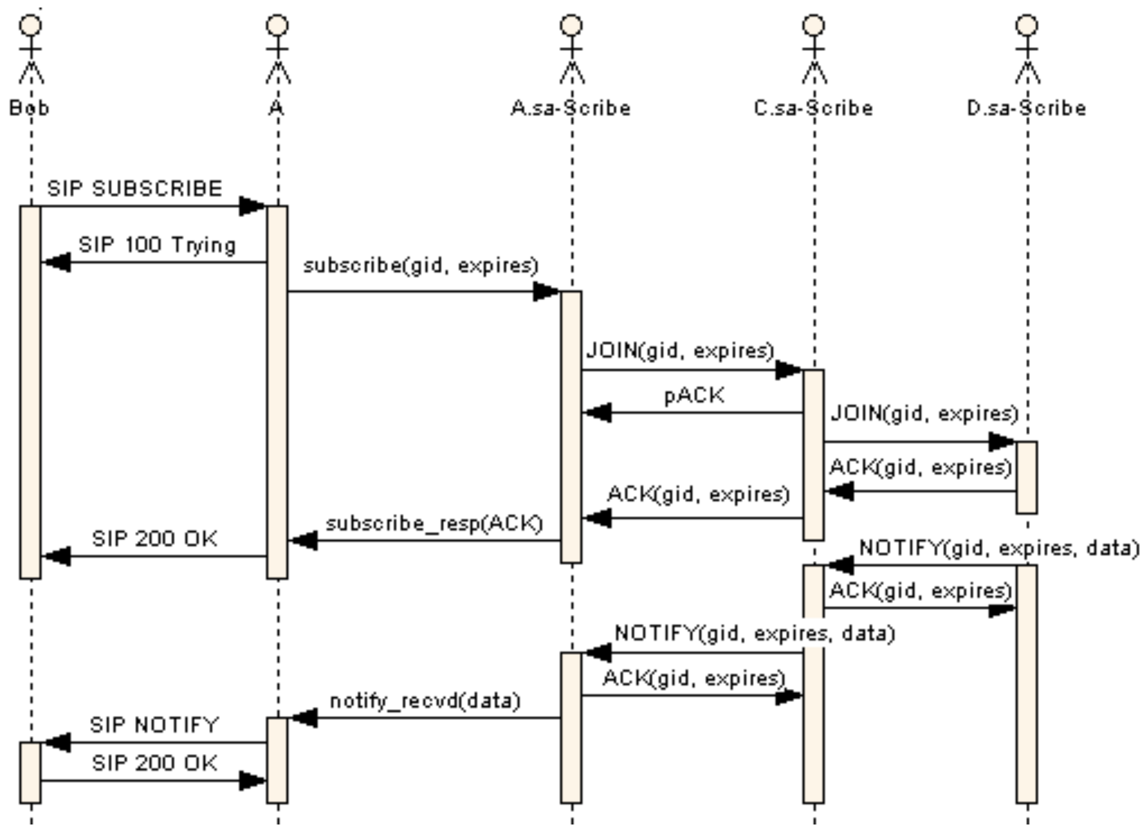


Figure 3.5: Subscribing for event notifications through the P2PSIP overlay

Up to this point we have assumed that the gid is the outcome of the overlay hash function on the SIP Request URI. In a real implementation of the proposed architecture this calculation strategy is insufficient. This is because the event state notification and publication mechanisms of SIP are abstract (template) mechanisms that in practice are used by many event packages (i.e. presence, xcap-diff, etc.). Thus, two SIP SUBSCRIBE requests for the same Request URI but with different Event type headers are unique subscriptions that require different notification data. To overcome this challenge the SIP event header information may be used in the calculation of the gid as well. This results in a more fine grained distribution of notification groups among the overlay peers, as the multicast root now serves notifications coupled to a particular event package type.

3.3.4 Failure and recovery

Failure and recovery is effectively handled by Scribe. A parent in the multicast tree sends periodic heartbeat messages to its children. If a child fails to receive heartbeat messages from its parent, then it re-initiates the JOIN procedure towards gid. The JOIN message is routed through the overlay and the multicast tree is repaired [35].

The sa-Scribe extends this functionality and modifies the JOIN message to carry the latest known state maintained by a child. Thus, the new parent in the multicast tree (eventually the new multicast root) becomes aware of the latest known notification state of the notification group. Notice that the new multicast root uses the sequence numbers to identify the latest event state, if JOIN requests from different children with different event state arrive.

An upcoming issue is the replication of publication related information that is only kept at the multicast root (i.e. etag and expiration timers). Scribe proposes a solution to this problem by replicating the state associated with the multicast root to the roots K nearest peers. Thus, upon a multicast root failure the multicast root group identifiers will be handled by one or more of these replication points that will act on behalf of the failed multicast root as its replacement peers.

Chapter 4
Evaluation

This Chapter investigates the performance of the proposed architecture when the overlay is destabilized (i.e., under churn). The base component of the notification architecture is a structured overlay network and on top of it the whole notification architecture is structured, since both the multicast and wrapper layers exploit the routing functionality provided by the overlay to achieve their operations. When no churns occurs in the overlay, the P2P network is said to be stabilized, since all the participating peers maintain up to date information in their routing tables. In this case the quality of service guarantees of the P2P component are met and the overlay is able, with high probability, to route a message with just $\lceil \log N \rceil$ lookups. However, when the overlay is under churn, as new peers enter the overlay and existing peers fail at random, the routing tables of the overlay peers contain inconsistencies that lead to increased lookup times, thus affecting both the message delivery rate and time.

The fact that structured peer overlays are affected by the presence of churn is thoroughly discussed in the Background Sections 2.2 and 2.3. Since no common remedy exists for completely mitigating the effect of churn in a structured P2P system, a rule of thumb states that the proper configuration deeply affects the observed performance. Relevant research conducted in [52, 49, 55, 53] proposes a mix of techniques that can partially mitigate the effect of churn but not address it in a full extent. These practices focus on fine tuning the overlay parameters according to the deployment environment. Thus, parameters as the soft state timers, the size of the routing table, as well as the neighbor set, all have an influence in the final overlay behavior. More recent research efforts in [52], as well as discussions available in the mailing list of P2P-SIP end up proposing peer overlays that reconfigure on real time, based on the experienced churn load.

Since our architecture is targeted to peers with mobile characteristics and volatile nature, it is of interest to evaluate the performance of a structured P2P system when the churn occurs and thus, we set up a simulation based on the P2P simulator Oversim [3]. We perform various simulation runs by varying the churn load, where each of the peers participates in an event notification service offered by the ALM protocol. During the simulations we measure the routing performance of the overlay expressed in number of lookups required to route a key in the overlay. More specifically, we measure the average lengths of the distribution trees created by the multicast protocol, as the length of the multicast tree is directly correlated with the lookup functionality delivered by the overlay.

4.1 Simulation engine

The Oversim comes as a plug-in of the well known simulation framework Omnet++ [16] and provides a modular architecture consisting of an underlay, overlay, and various application layers. The modular architecture of Oversim is shown in Fig.1.7, Section 1.4 and is further depicted here for easier understanding.

According to Fig.4.1, the base component in Oversim architecture is the underlay, which simulates the IP

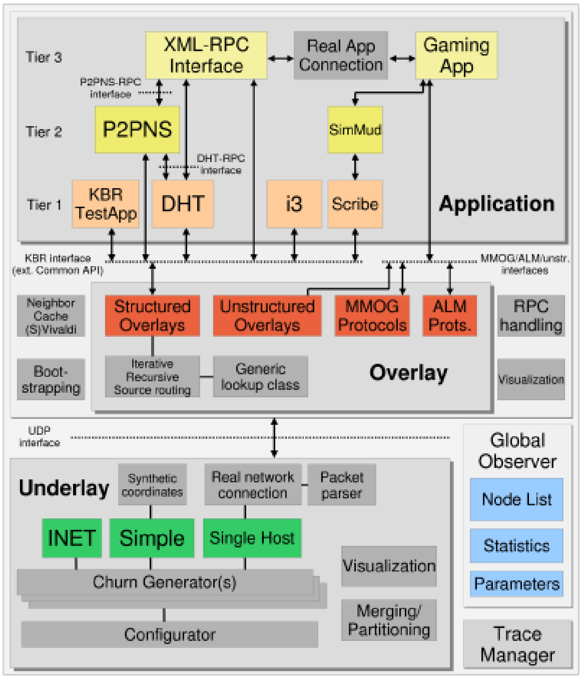


Figure 4.1: The Oversim architecture [3]

connectivity among the peers. The underlay component can be configured to simulate a full scale IPv4 or IPv6 network through the “INET” plugin of the Omnet++, or it can be tuned to accommodate simpler networking topologies through the “Simple” plugin. Moreover, if the “Single host” underlay is selected then instead of simulation the Oversim operates in emulation mode. In our case the underlay is configured to the ”Simple” mode, where the simulator creates a simplified network access scheme based on unique identifiers instead of IP addresses, as well as a UDP-like transport protocol, with minimal functionality. With this underlay configuration, which is also known as “flat topology”, the delays between the hosts are not created based on a topology aware metric (i.e. Euclidian distance), instead the simulator assign random or fixed delays between the simulated peers. We chose to represent the delays between the hosts with a fixed metric, where every peer can reach its immediate neighbors with a constant delay (i.e. 50ms). Choosing a fixed delay among the peer links does not affect the generality of the measurements that are primarily targeted on the mean number of lookups required to route a packet when the churn load varies in the overlay. However, we are aware that his approximation in the introduced delay between the peers is generally not the case, since delays vary according to peer’s mobility, network characteristics, network distance, etc.

On top of the underlay component resides the overlay component, which hosts different structured overlay algorithms. In our simulations we chose the Pastry algorithm as it performs better with the Scribe ALM scheme, see Subsection 2.4.2, which was configured to run as an application layer protocol in the simulations. The modularity of Oversim further organizes the application layer into two tiers, thus, it enables a second application to run in tier-2 if another application already operates on tier-1. In our simulations we developed a “driver” application operating on top of the Scribe protocol just above the Pastry overlay algorithm. This tier-2 application was used for stimulating the simulation, as well as for data gathering and processing.

4.2 Simulation set-up

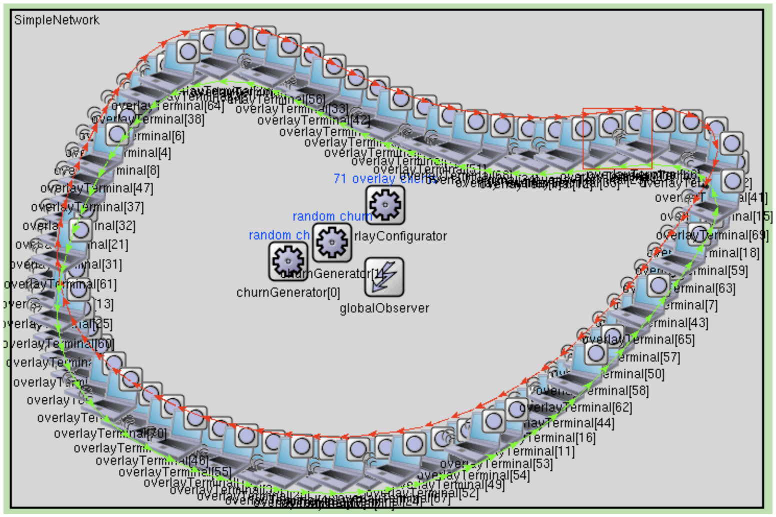


Figure 4.2: A snapshot of the simulated overlay; for rendering purposes the depicted topology holds 71 peers, compared to the actual one that holds 1001

The set up of the simulations involved 1000 volatile consumer peers and one static publisher peer. The publisher peer is generated by Oversim. The "driver" application running on this peer invokes a *create(gid)* call at the Scribe protocol, providing as *gid* its own routable peer address. Thus, the publisher acts as the multicast root for its own publications. After the publisher is created and initialized, the peer-generator of Oversim generates the other 1000 subscriber peers. As soon as a subscriber peer instantiates, the "driver" application forces it to subscribe through the multicast call *join(gid)*, providing as *gid* the overlay address of the publisher. By the time all the subscriber peers are generated and instantiated, the publisher starts publishing information at regular intervals of $5sec$ and some of the subscribers start to fail and return into the overlay with a different transport underlay address. The publication interval, does not affect the generality of the measurements since in the end we express our results using sample mean. The peer migration action is similar to a peer leaving the overlay, without first notifying the overlay and substituted by a new peer that enters the overlay with no routing information. Thus, when varying the migration rate from 2 to 30 peers per minute, we introduce different degree of churn into the overlay and trigger its stabilization routines in order to keep the routing tables of the remaining peers up to date with the changes in the overlay structure.

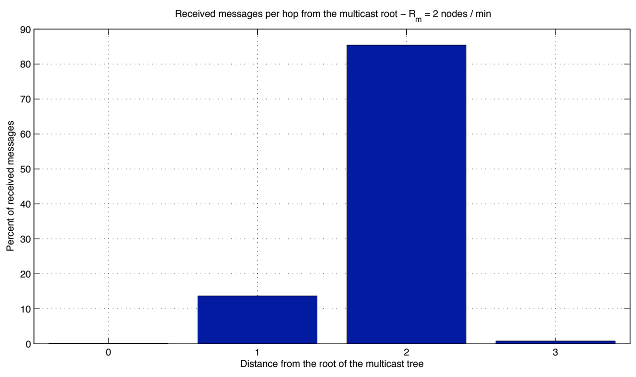
Since, the outcome of the results for the simulation set up can be biased from many sources, we chose to set up both the overlay as well as the ALM protocol in a churn aware configuration. More specifically, the Scribe, as well as the Pastry, integrity timers were tuned at $2sec$, while the migration rate varied in each of the simulation runs with values 2, 5, 10, 30 migrating peers/minute, accordingly. A view of the simulated P2P ring, as captured by Oversim is given in Fig.4.2.

In order to monitor the length experienced by each multicasted data packet, the "driver" application of the publisher peer is configured to insert a hop count field in the payload of the transmitted notification packets. As soon as a new publication is generated it is communicated down the multicast tree, till it reaches the leaf consumer peers. Every intermediate peer between the multicast root and the leafs prior to forwarding the packet down the tree, increases the hop count of the data packet and keeps track of the reception time for that particular notification. Finally, the "driver" application in each of the subscribers collects the hop count statistics per received notification into a common logging facility. The distribution of the average tree length experienced by the overlay peers when receiving the notification messages are depicted below, and further summarized in the following Table:

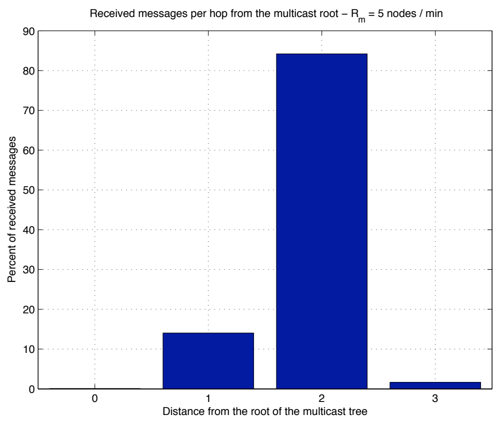
Migration rate (peers/minute)	Error rate (lost messages)	Max tree length (number of hops)
0	0	$3 = \lceil \log_{16} 1001 \rceil$
2	2.1	3
5	5.4	3
10	10.4	4
30	27.65	10

Table 4.1: Accumulative results (sample mean values) regarding error rate and max tree length observed, when running the simulations for [0, 2, 5, 10, 30] migrating peers per minute. The increased error rate appears because of the increased migration rate. At this case, Scribe cannot maintain the integrity of the tree. Notice that Scribe does unreliable delivery of notifications. In case a simply acknowledgment model is followed, like the one of sa-Scribe, then reliable delivery of notifications can be achieved
.

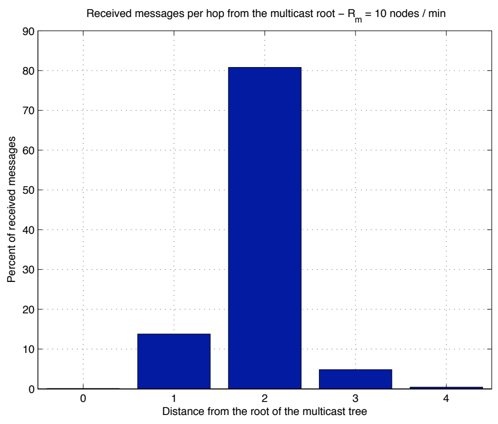
The results depicted in the Figures 4.3a, 4.3b, 4.3c, 4.3d, and further summarized in Table 4.1 show that when the overlay operates under churn, then service discontinuities are experienced. Moreover, the routing performance of the overlay maintain good characteristics in every churn region, from low (2, 5 migration actions per minute) and medium (10) to high (30). Especially in the region of medium churn where the migration rate set to 10 peers per minute, the average tree length experienced by the packets was only one hop more (4) than the bounded behavior experienced in a stabilized environment. Moreover, in the case were the churn load reached 30 migrating peers per minute; even though the maximum tree length recorded reached up to the 7 hops, the majority of the peers (ca 80%) still received their messages with only two hops distance from the multicast root. This behavior shows that even when high churn occur within the overlay, a reliable transmission scheme, can deliver packets without excessive delays.



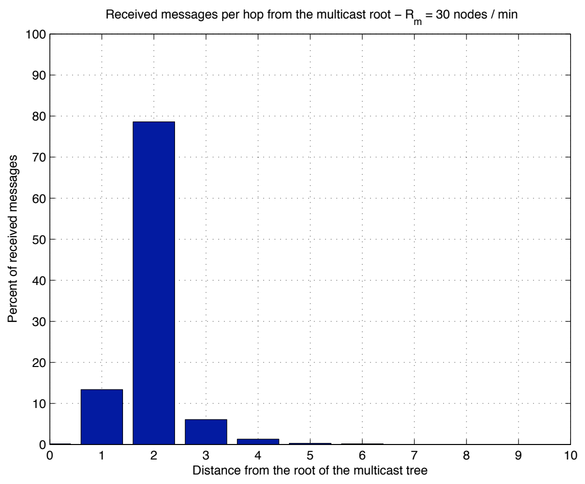
(a)



(b)



(c)



(d)

Figure 4.3: The distribution of the average tree length experienced by the overlay peers when receiving the notification messages. Migration rate is 2 (a), 5 (b), 10 (c), 30 (d) peers per minute

Chapter 5
Conclusions and future work

This thesis presents a decentralized event notification architecture intended to be used by the EU FP6 project MUSIC, as a context exchange mechanism. The proposed notification architecture leverages a structured P2P overlay algorithm and an ALM protocol to achieve decentralization of the notification load within the peer overlay. The proposed architecture contributes to the P2PSIP efforts and enables the SIP/SIMPLE presence package to support distributed and decentralized event notifications. Moreover, the architecture is transparent to the end users that access the notification overlay using the SIP standards for event notifications and event state publications.

The initial architecture design focused on openness, modularity and re-configurability. The primary objective is to use the architecture components off-the-shelve. This produces an agile and extensible architecture, since the P2P overlay algorithm and ALM protocol are accessed through a collection of public and open APIs that allow the architecture to operate with different component realizations. Coordination between the components is achieved through a custom "wrapper" layer. However, this design approach mandates that the custom protocol as well as the ALM protocol create and maintain the same path to disseminate notification requests. As a result the architecture installs extra computational and network overhead to the overlay peers, as exactly the same path state needs to be maintained in two protocols running on the same node. In addition, the decision to adapt the reverse path forwarding technique by the wrapper "protocol" violate the extensibility objective as non every ALM protocol implementation can be plugged into the architecture. Since the initial architecture design could not be met we focused on a second, alternative, design targeting operational and implementation simplicity. The second design leverages a custom ALM protocol - SIP aware Scribe (saScribe) - , that is based on Scribe [35], extended to support SIP like notifications behavior.

Moreover, we are evaluating the ancestor of saScribe, the Scribe ALM protocol in a mobile environment where the peers disconnect and join the overlay at random intervals. The simulation results show that the architecture can be deployed in environments with low or moderate peer mobility, as long as a reliable transport channel is used to recover the lost messages (due to temporarily broken multicast trees). The extensions proposed for saScribe provide a simple reliability mechanism based on the use of acknowledgements that solves this problem. However, the evaluation of the architecture only provides some initial proof of concept behavior and an elaborate study of the achieved scalability is currently work in progress. Finally, the advances of the P2PSIP working group are of interest and an internet draft describing the proposed notification architecture is in our future plans.

References

[1] A. Niemi. Session Initiation Protocol (SIP) Extension for Event State Publication. RFC 3903, Internet Engineering Task Force, October 2004.

[2] A. B. Roach. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265, Internet Engineering Task Force, June 2002.

[3] Ingmar Baumgart, Bernhard Heep, Sthephan Krause, and Sebastian Mies. Oversim: The overlay simulation framework.

[4] D. Bryan, P. Matthews, E. Shim, and D. Willis. Concepts and Terminology for Peer to Peer SIP. Internet-Draft draft-ietf-p2psip-concepts-01, Internet Engineering Task Force, November 2007. Work in progress.

[5] C. Jennings, B. Lowekamp, E. Rescorla, J. Rosenberg, S. Baset, and H. Schulzrinne. REsource LOcation And Discovery (RELOAD). Internet-Draft draft-bryan-p2psip-reload-03, Internet Engineering Task Force, 2008. Work in progress.

[6] B Schilit, N Adams, and R Want. Context-aware computing applications. In *in IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.

[7] Anind Dey and Gregory Abowd. Towards a better understanding of context and context-awareness. Technical report, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, 1999.

[8] W. Li. *Towards a Person-Centric Context Aware System.* Licentiate thesis, Department of Computer and System Services, Stockholm University and Royal Institute of Technology, Kista, Sweden, 2006.

[9] T. Ernst, M. Montavont, R. Wakikawa, C. Ng, and K. Kuladinithi. Motivations and Scenarios for Using Multiple Interfaces and Global Addresses. Internet-Draft draft-ietf-monami6-multihoming-motivation-scenario-03, Internet Engineering Task Force, May 2008. Work in progress.

[10] D. Pave and D. Trossen. Context Provisioning and SIP Events in Workshop on Context Awareness at ACM SIGMOBILE, 2004.

[11] Carlos Angeles Piña. Distribution of Context Information using the Session Initiation Protocol (SIP). Master's thesis, School of Information and Communication Technology, The Royal Institute of Technology (KTH), June 2008.

[12] IST project 035166. Self-adapting applications for mobile users in ubiquitous computing environment project.

[13] The MUSIC Consortium. D4.2 system design of the music architecture. Internet, 2008.

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[15] RATP - Transports en Île-de-France - Metro map.

[16] András Varga. OMNeT++ Discrete Event Simulation System.

[17] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force, January 2005.

[18] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. Internet-Draft draft-ietf-mmusic-ice-19, Internet Engineering Task Force, October 2007. Work in progress.

[19] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, Internet Engineering Task Force, March 2003.

[20] J. Rosenberg, R. Mahy, and P. Matthews. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). Internet-Draft draft-ietf-behave-turn-07, Internet Engineering Task Force, February 2008. Work in progress.

[21] J. Rosenberg. SIMPLE made Simple: An Overview of the IETF Specifications for Instant Messaging and Presence using the Session Initiation Protocol (SIP). Internet-Draft draft-ietf-simple-simple-02, Internet Engineering Task Force, February 2008. Work in progress.

[22] P2P-SIP Working Group.

[23] Vishal K. Singh and Henning Schulzrinne. Simplestone - benchmarking presence server performance. Technical report, Columbia University, 2004.

[24] A Carzaniga, D S Rosenblum, and A L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, (19):200–1, 2001.

[25] A Gupta, O D Sahin, D Agrawal, and A E Abbadi. Meghdoot: Content-based publish/subscribe over p2p networks. In *In Middleware*, pages 254–273. Springer, 2004.

[26] P R Pietzuch and J Bacon. Peer-to-peer overlay broker networks in an event-based middleware. In *In Proc. of the 2nd Int. Workshop on Distributed Event-Based Systems*, pages 1–8. ACM, 2003.

[27] D Tam, R Azimi, and H-A Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. pages 138–152, 2003.

[28] Ioannis Aekaterinidis and Peter Triantafillou. Pastrystrings: Content-based publish/subscribe over dht networks.

[29] V Muthusamy and H-A Jacobsen. Small-scale peer-to-peer publish/subscribe. In *In P2P Knowledge Management Workshop at MobiQuitous*, 2005.

[30] Stephen E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[31] D. R. Cheriton and S. E. Deering. Multicast routing in datagram internetworks and extended LANs. *ACM Trans. on Computing Sys.*, 8(2):85, May 1990.

[32] Stephen E. Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. The PIM architecture for wide-area multicast routing. *IEEE/ACM Trans. Netw*, 4(2):153–162, 1996.

[33] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *SIGCOMM*, pages 205–217. ACM, 2002.

[34] Ying Zhu, Baochun Li, and Jiang Guo. Multicast with network coding in application-layer overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1):107–120, 2004.

[35] Miguel Castro, Peter Druschel, Anne marie Kermarrec, and Antony Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure, September 01 2002.

[36] Shelley Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV*, pages 11–20. ACM, 2001.

[37] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level multicast using content-addressable networks. *Lecture Notes in Computer Science*, 2233:14–??, 2001.

[38] B. Cohen. The BitTorrent protocol specification.

[39] J Guterman. Gnutella to the rescue ? not so fast, napster fiends. link to article at http://gnutella.wego.com, 2000.

[40] National Institute of Standards and Technology (NIST). Secure Hash Standard. Internet, August 2002.

[41] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. 2001.

[42] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[43] Antony Rowstron and Peter Druschel. Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[44] American National Standard. *LAN: 802.5, Token Ring*. 1985.

[45] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Theory of Computing Systems*, 1999.

[46] P MAYMOUNKOV and D MAZIERES. Kademlia: A peer-to-peer information system based on the xor metric. 2002.

[47] R Bhagwan, S Savage, and G Voelker. Understanding availability. In *In Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS*, 2003.

[48] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell, and Seif Haridi. Comparing maintenance strategies for overlays. *CoRR*, abs/0710.0386, 2007.

[49] J Li, J Stribling, T M Gil, R Morris, and M F Kaashoek. Comparing the performance of distributed hash tables under churn. In *In Proc. IPTPS*, 2004.

[50] B Y Zhao, J D Kubiatowicz, and A D Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, 2001.

[51] I Gupta, K Birman, P Linga, A Demers, and R van. Renesse. kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *In IPTPS*, 2003.

[52] Supriya Krishnamurthy, Sameh El-Ansary, Erik Aurell, and Seif Haridi. An analytical study of a structured overlay in the presence of dynamic membership. *CoRR*, abs/0710.0270, 2007.

[53] S Rhea, D Geels, T Roscoe, and J Kubiatowicz. Handling churn in a dht. Technical report, 2004 USENIX Annual Technical Conference, June-July, 2003.

[54] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988, Internet Engineering Task Force, November 2000.

[55] M Castro, M Costa, and A Rowstron. Performance and dependability of structured peer-to-peer overlays. Technical report, 2003.

[56] D. Waitzman, C. Partridge, and S.E. Deering. Distance Vector Multicast Routing Protocol. RFC 1075, Internet Engineering Task Force, November 1988.

[57] J. Moy. MOSPF: Analysis and Experience. RFC 1585, Internet Engineering Task Force, March 1994.

[58] B. Fenner, H. He, B. Haberman, and H. Sandick. Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding (IGMP/MLD Proxying). RFC 4605, Internet Engineering Task Force, August 2006.

[59] X Zhang, J Liu, B Li, and T-S P Yum. Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming. In *In IEEE Infocom*, 2005.

[60] Rongmei Zhang and Y. Charlie Hu. Borg: a hybrid protocol for scalable application-level multicast in peer-to-peer networks. In *NOSSDAV*, pages 172–179, 2003.

[61] Zi Chu, Yin yin Yang, Hao Tu, and Wei ning Kong. Ubicast: An enhanced application-level multicast protocol. In *PDCAT*, pages 762–767, 2005.

[62] Y K Dalal and R M Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, (21):1040–1048, 1978.

[63] Miguel Castro, Michael B. Jones, Anne marie Kermarrec, Antony Rowstron, Marvin Theimer, and Helen Wang. An evaluation of scalable application-level multicast built using peer-to-peer overlays, December 27 2003.

[64] David A. Bryan, Bruce Lowekamp, and Cullen Jennings. SOSIMPLE: A serverless, standards-based, P2P SIP communication system. In *AAA-IDEA*, pages 42–49. IEEE Computer Society, 2005.

[65] Erkki Harjula, Jussi Ala-Kurikka, Douglas Howie, and Mika Ylianttila. Analysis of peer-to-peer SIP in a distributed mobile middleware system. In *GLOBECOM*. IEEE, 2006.

[66] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Internet Engineering Task Force, April 2006.

[67] E. Rescorla and N. Modadugu. Datagram Transport Layer Security. RFC 4347, Internet Engineering Task Force, April 2006.

[68] Frank Dabek, Ben Zhao, Peter Druschel, and John Kubiatowicz. Towards a common api for structured peer-to-peer overlays.

[69] Nodoka Mimura, Kiyohide Nakauchi, Hiroyuki Morikawa, and Tomonori Aoyama. A middleware approach for supporting application-level multicast services.