Sarantorn Bisalbutra

Publish/Subscribe Gateway for Real-time Communication

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.
Espoo 19.11.2012

Thesis supervisor:

Prof. Jörg Ott
Aalto University

Thesis instructor:

M.Sc. (Tech.) Petri Jokela
Oy L M Ericsson Ab

Thesis instructor:

M.Sc. (Tech.) Jimmy Kjällman
Oy L M Ericsson Ab

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Sarantorn Bisalbutra

Title: Publish/Subscribe Gateway for Real-time Communication

Date: 19.11.2012 Language: English Number of pages: 13 + 83

Department of Communications and Networking

Professorship: Networking Technology Code: S-38

Supervisor: Prof. Jörg Ott, Aalto University

Instructor: M.Sc. (Tech.) Petri Jokela, L M Ericsson

Instructor: M.Sc. (Tech.) Jimmy Kjällman, L M Ericsson

This thesis proposes a design of a gateway, which connects IP and publish/subscribe networks together, enabling their co-existence, for example, during an IP to pub/sub migration phase.

There is a proposal to revise the architecture of the present Internet, from "*Host-Centric Networking*" to a new concept called "*Information-Centric Networking* (ICN)". One of the ongoing projects in this field is the PURSUIT project, which uses the publish/subscribe paradigm as a basic communication model. Since the proposal from the PURSUIT project has gained quite much interest recently, the next step is to consider the process of deploying the new Internet architecture. This thesis focuses on gateway's mechanism to transparently convert IP-based end-to-end traffic to the publish/subscribe based and vice versa, in order to support voice communication using Session Initiation Protocol as well as multimedia streaming over multicast.

The main idea of our design is to allow operators to utilize the features of Information-Centric Networking, while home users or companies can still use legacy IP connectivity and applications. In this scenario, the operators will gain benefits from new solutions, e.g., stateless Bloom-filter based multicast forwarding in the pub/sub network. We describe the gateway's functionalities to handle SIP session initialization, parameters negotiation, media session establishment, as well as maintaining and terminating the session over the publish/subscribe network. This includes a design of a pub/sub based SIP registrar for taking care of user registration, call redirection, and mobility. Moreover, we also discuss the mechanism to support multimedia streaming over multicast. Our gateway is responsible for group establishment, session joining and leaving, and eventually group termination.

In addition to our design, we describe an implemented prototype, and evaluate the system's functionalities according to the requirements of this thesis. After that, we analyze the performance of the design and implementation, traffic density during different phases of both SIP and multicast sessions, and finally compare the call setup duration between IP and pub/sub networks.

Keywords: Information-Centric Network, publish/subscribe gateway, pub/sub SIP registrar

AALTO-YLIOPISTO DIPLOMITYÖN
SÄHKÖTEKNIIKAN KORKEAKOULU TIIVISTELMÄ

Tekijä: Sarantorn Bisalbutra

Työn nimi: Julkaisu/tilaus-yhdyskäytävä reaaliaikaista kommunikaatiota varten

Päivämäärä: 19.11.2012 Kieli: Englanti Sivumäärä: 13 + 83

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkkotekniikka Koodi: S-38

Valvoja: Prof. Jörg Ott, Aalto-yliopisto

Ohjaaja: M.Sc. (Tech.) Petri Jokela, L M Ericsson

Ohjaaja: M.Sc. (Tech.) Jimmy Kjällman, L M Ericsson

Tässä diplomityössä esitellään yhdyskäytävä, jolla voidaan yhdistää IP-verkot ja informaatiokeskeiset julkaisu/tilaus-verkot toisiinsa sekä mahdollistaa niiden käyttäminen rinnakkain.

Internetin arkkitehtuuria on ehdotettu uudistettavaksi siten, että nykyisestä ns. isäntäkeskeisestä mallista siirryttäisiin informaatiokeskeiseen malliin. Eräs projekti, jossa tätä tutkitaan, on PURSUIT, jossa tietoliikenne perustuu julkaisu/tilaus-malliin. Projektissa on otettu huomioon myös tämän uuden arkkitehtuurin käyttöönottaminen Internetissä. Tähän liittyen tässä diplomityössä on suunniteltu yhdyskäytävä, joka muuntaa IP-liikenteen julkaisu/tilaaja-pohjaiseksi ja päinvastoin. Muunnos voidaan tehdä SIP-protokollaa käyttäville puheluille sekä IP-monilähetystä hyödyntäville multimediavirroille.

Yhdyskäytävän avulla operaattorit voivat hyödyntää verkossaan informaatiokeskeisen mallin ominaisuuksia sekä siihen liittyviä mekanismeja, kuten tilatonta monilähetystä, ja verkon käyttäjät puolestaan voivat edelleen käyttää IP-yhteyksiä ja -sovelluksia. Työssä kuvataan, yhdyskäytävän toiminnallisuudet, jotka mahdollistavat SIP-istunnon alullepanemisen, parametrien neuvottelun, media-istunnon käynnistämisen sekä istunnon ylläpitämisen ja katkaisemisen julkaisu/tilaus-verkon ylitse. Työssä on myös suunniteltu SIP-rekisteriöintipalvelinsovellus, joka hoitaa käyttäjien rekisteröinnin, puheluiden uudelleenohjaukset sekä käyttäjien liikkuvuuden. Lisäksi kuvataan yhdyskäytävään sisältyvä mekanismi, jolla multimedian virtauttaminen monilähetyksenä on toteutettu. Yhteyskäytävä vastaa tässä tapauksessa monilähetysryhmien luomisesta ja purkamisesta sekä istuntoihin liittymisestä ja poistumisesta.

Yhdyskäytävän suunnitelman lisäksi diplomityössä kuvataan prototyypin toteutus sekä arvioimme järjestelmän vastaavuutta työssä määriteltyihin vaatimuksiin. Lisäksi analysoimme järjestelmän suorituskykyä ja liikenteen määrää istuntojen eri vaiheissa, sekä vertaamme näitä tuloksia IP- ja julkaisu/tilaus-verkkojen välillä.

Avainsanat: julkaisu/tilaus-yhdyskäytävä, informaatiokeskeinen verkko, SIP-rekisteröinitipalvelin

Acknowledgement

First of all, I would like to thank Professor Jörg Ott for recommending me to apply for this thesis opportunity in Nomadiclab, Ericsson Research Finland, and also for being my supervisor. I really appreciate his comments and suggestions during the period of this thesis.

Secondly, I would like to show my appreciation to Petri Jokela for giving me the opportunity to work in PURSUIT project, as well as being my instructor in Nomadiclab, for the guidance, very useful comments, feedbacks and valuable work experience. Also, I would like to thank Jimmy Kjällman, my other instructor, for all the supports, especially with the Blackadder prototype. Thank you very much for participating in my work since the very beginning, clarifying all the questions, helping and advising me, particularly during the design and implementation phase. Again, I would like to thank both of them, for helping me to scope the work, for all their time and patience, especially during the writing phase. I feel very grateful, and I would like to say that this is one of the best and most motivating experiences of mine.

In addition, I would like to thank all my colleagues in Nomadiclab for the very friendly and joyful environment, especially Ari Keränen and Teemu Rinta-aho for being my additional interviewers, Tony Jokikyyny for being a very kind and understanding manager, and Jukka Ylitalo for answering my questions.

Finally, I would like to show the gratitude to my family, and friends for all their supports. Thank you so much for listening and cheering me up anytime I have problems or worries.

Jorvas, November 19, 2012

Sarantorn Bisalbutra

Contents

Abbreviations and Acronyms

AOR	Address of Record
API	Application Programming Interface
DNS	Domain Name System
FID	Forwarding Identifier
FTP	File Transfer Protocol
FW	Forwarding node
FWD	Forwarding System
IANA	Internet Assign Number Authority
ID	Identifier
IGMP	Internet Group Management Protocol
INT	Invite
IP	Internet protocol
LID	Link Identifier
NID	Node Identifier
NTP	Network Time Protocol
PDU	Protocol Data Unit
Pub/sub	Publish/subscribe
REQ	Request
RES	Response
RID	Rendezvous Identifier or Information Identifier
RTP	Real-time Transport Protocol
RTCP	RTP Control Protocol
RV	Rendezvous
SAP	Session Announcement Protocol
SDP	Session Description Protocol
SID	Scope Identifier
SIP	Session Initiation Protocol
TM	Topology Manager
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Uniform Resource Identifier

List of Figures

List of Tables

Chapter 1

Introduction

As the usage of the Internet has changed and become more diverse, comparing to the purpose of its design in the 1970s [16], proposals to change and improve its architecture have come up. One project, doing research in this area is the Publish/Subscribe Internet Technology (PURSUIT). It proposes a new communication scheme based on *Information-Centric Networking* and the *publish/subscribe* (pub/sub) paradigm [43]. This thesis is written within the context of the PURSUIT project, presenting the design and implementation of a gateway between the current Internet and the PURSUIT publish/subscribe networks, enabling the cooperation among the two.

From 1970 the Internet has been used to provide communications between people and machines [53] based on a send and receive model. The *Internet Protocol* (IP) works by identifying and locating each network device's interface with an IP address, and the network takes a role to route data packets according to those destination addresses. This approach is called *Host-Centric Internetworking*.

Since the Internet has been used worldwide, it has been integrated into many aspects of people's lives including the society, economics, and politics [11], becoming one of the most important communication infrastructures in the world. The number of Internet users has increased dramatically. Also, vast amount of applications have been developed to provide multiple types of services, including World Wide Web, Voice over Internet, Video calls, and Peer-to-Peer File Sharing.

However, in the beginning of the Internet era, information about the source node was obvious. The user had a clear viewpoint about where the information came from, but the situation in the current Internet model is different. Technological advancement and new applications introduce disassociation between the data content and location of the resources. The same piece of data can come from multiple hosts inside the network, although the application is pointing to a single host name. An example of these applications is the web browsing. The users make a request to only one web URL, whereas the web page might possibly come from (1) any HTTP server in the network, (2) the HTTP proxy, or (3) a caching system. This indicates that the information turns out to be more important than the data source in the users perspective. Moreover, users nowadays tend to primarily concern about the speed of data retrieving than any other aspects, as we could see in the peer-to-peer file sharing service (BitTorrent [10]). This contributed to a new concept of the design called *Information-Centric Networking* (ICN).

Research works on ICN have been receiving attention over the past few years. The PURSUIT project, using publish/subscribe paradigm as a basic communication model, is one of them. The idea of PURSUIT publish/subscribe internetworking is based on publication and subscription of information in the network. When a *publisher* wants to make a piece of data available, it publishes the meta-information about this data to the network.

On the other hand, a *subscriber* will issue the subscription, when the data is needed. These data publications and subscriptions are structured and stored inside the network, in a so-called *Information Structure,* indicating overall figure of the existence and the need of each specific information item. Finally, when there is a matching between the publication and subscription pair, the data will be forwarded to the subscriber.

In the PURSUIT architecture, the publisher is responsible for publishing updated pieces of information. The subscriber does not usually have to make any request for the new information, because the data is delivered according to the subscription. Furthermore, this type of pub/sub network provides better security with multiple layers of access privileges [13].

Since the PURSUIT publish/subscribe paradigm has recently gained interest, the next step is to consider the process of deploying the PURSUIT's publish/subscribe network as a new Internet architecture. However, this cannot be done overnight. A gateway between IP and publish/subscribe networks is one possible approach to enable their co-existence, during the migration phase. The benefit of this gateway is, for instance, to allow operators to utilize the new features of Information-Centric Networking and related technologies, while home users or companies can still use legacy IP networks and applications.

This thesis describes the design and presents the result of the gateway implementation. The assumption is that the operator network utilizes PURSUIT publish/subscribe paradigm, while the customer networks are using legacy IP routing mechanism. The locations of the pub/sub gateways are designed to be located at the connecting point between these two types of networks, and seamlessly provide suitable packet conversion between them. More details about the goal and scope of this thesis work are described in the following subsections.

1.1 Goal and Scope of the Thesis

The gateway implementation consists of three major tasks:

1. The gateway is required to convert IP-based end-to-end traffic to publish/subscribe based and vice versa. In our scenario, both higher level signaling and lower level packet routing have to be converted at the gateway. Besides, the gateway needs to support both unicast and multicast techniques at the same time.
2. Session Initiation Protocol (SIP) will be used as an application layer protocol. The clients at both ends of the connection are using IP-based SIP applications, while the publish/subscribe network takes care of connection setup and packet forwarding between them. The gateway initializes the session, takes care of parameter negotiation between each peer in the IP network, establishes RTP connection between end hosts and maintains the connectivity throughout the period of the connection.

3. The benefit of PURSUIT pub/sub multicasting is demonstrated by using RTP multicast streaming. The IP multicast stream is generated from one IP designated side to several end users behind other gateways. The gateway handles multicast sessions proficiently including multicast initiation, group joining and leaving.

In this thesis, we implement a SIP service in the present Internet architecture, enabling the user mobility. To support this operation, an implementation of SIP registrar for the publish/subscribe network is necessary. This registrar is installed in the pub/sub network, utilizing PURSUIT publish/subscribe mechanism instead of the normal IP routing. It takes care of SIP user registration, SIP URI [57] and address binding as well as SIP destination discovery, which happen before the actual connection establishment.

Scope

This thesis focuses on the PURSUIT publish/subscribe architecture and compatibility with the Blackadder implementation within the project. As a result, compatibility to other information centric alternatives is out of scope. Moreover, the Session Initiation Protocol (SIP) is used as an application layer (signaling) protocol for session initiation, management and termination. Therefore, compatibility to other application layer protocols is not included.

Real-time Transport Protocol (RTP) is used as the transport protocol for real time media communication in this thesis. In addition, we assume that IP version 4 addresses are used in the customer network, and the Session Announcement Protocol (SAP) version 1 is always utilized to announce the information about multicast sessions.

1.2 Structure of the Thesis

This thesis comprises of 7 chapters. Chapter 2 presents the essential background of this thesis, together with previous related research works. We discuss the requirements of this thesis in Chapter 3. Chapter 4 describes the overall conceptual design of the pub/sub gateway and SIP registrar. After that, Chapter 5 will present the system implementation and explain how the gateway and registrar's modules are structured in detail. Computational workflow of the gateway will be illustrated in the Chapter 5 as well. Chapter 6 discusses how the system evaluation and measurement were arranged. Also, we present the result and analysis of them in this chapter. Finally, the overall outcome during the thesis is summarized in Chapter 7.

Chapter 2

Background

In this chapter, we present an overview of related work. First, we introduce PURSUIT technology, representing one solution of the Information Centric Internetworking. We utilize PURSUIT's publish/subscribe technology as a communication infrastructure. Within the IP context, we discuss the functionalities of Session Initiation Protocol and Real-time Transport Protocol, especially how they contribute to a real-time communication. After that, we compare Bloom filter based multicast of the PURSUIT architecture with the IP multicast technique. Finally, we present Domain Name System, Session Announcement Protocol, and RTP Control Protocol, which are other supplementary protocols that our gateway needs to understand.

2.1 PURSUIT Publish/Subscribe Internet Technology

Information-Centric Networking (ICN) is a recently proposed solution for the Internet advancement. The motivation of the ICN comes from the behavioural changes of the Internet users, since they are aware of the data content more than the location or where the data comes from.

Recently, several different solutions have been proposed for the information-driven framework, e.g. CCNx protocol [9] and SAIL [55]. This section will focus on PURSUIT project, which is one of the proposed solutions.

PURSUIT is an FP7 EU-project [56], which continues the work in the Publish-Subscribe Internet Routing Paradigm (PSIRP) [48]. One main goal of the PURSUIT is to propose the Internet architecture that conforms to the social structure, and adapts itself according to the changes. At this stage, the work in PURSUIT project focuses on information routing and forwarding, based on the publish/subscribe paradigm [43].

The implementation used in this thesis is based on the Blackadder prototype [23]. Blackadder is an implementation of the PURSUIT architecture, implemented based on the information-centric functional model in [13]. In the next subsections, we go into more elaborate details, seeing how many subsystems are there in the Blackadder prototype, and how they work to provide efficient services.

2.1.1 Networking Model

Data exchange in the PURSUIT architecture is based on data publication and subscription. An example of the pub/sub network is shown in Figure 2.1. The most important components in this network are *Publishers*, *Subscribers*, *Rendezvous System (RV), Topology Manager (TM)* and *Forwarding nodes (FW)*, as shown in the figure.

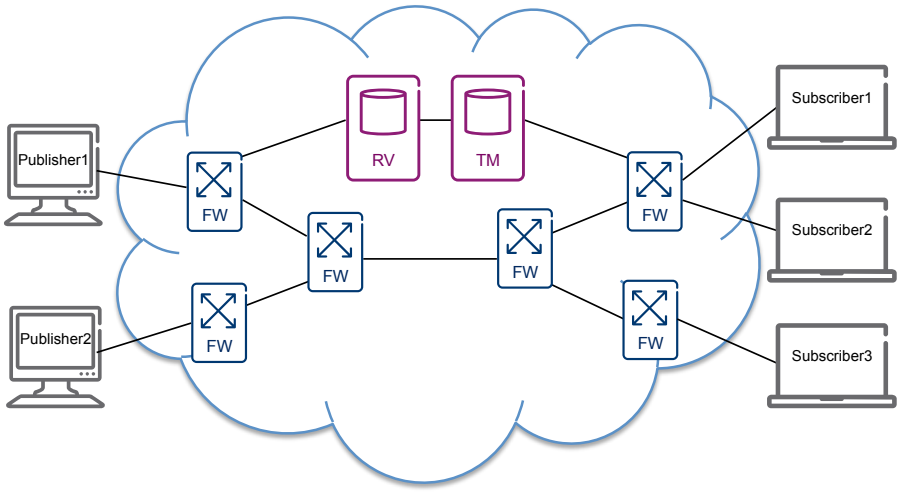


Figure 2.1: Example of PURSUIT domain

Figure 2.2 summarizes basic communication model of the PURSUIT architecture. The *publisher* is an entity, holding actual information pieces. It publishes this information by indicating an existence of the item to the network. Once the indication reaches the *rendezvous system*, it adds metadata of the item in a hierarchical tree, called an *information structure*.

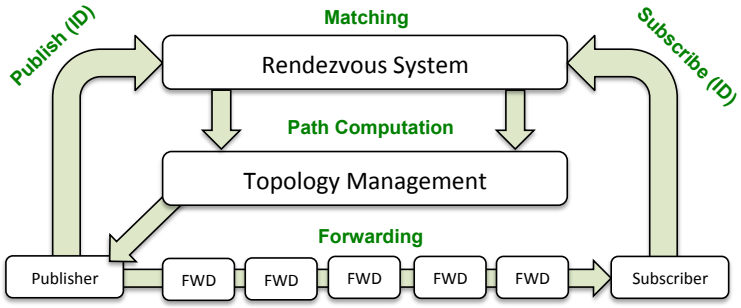


Figure 2.2: Example of PURSUIT communication model

Before publishing the information of each item, the publisher can publish the scope of information, combining a group of similar items together. This scope publication also reaches the rendezvous system, and the rendezvous takes a responsibility to add it into the tree. More detail about the information structure will be explained in Section 2.1.1.1.

The *subscriber* states its interest to receive the corresponding information by making a subscription to the scope or the information item. The rendezvous matches the subscription to the publication. If it notices that the information is available, the rendezvous node creates a signal which is forwarded to the *topology manager*, who takes care of the path calculation.

The topology manager has overall knowledge about the network, including node identifier (NID), number of links and the link identifiers (LID). The TM is responsible for calculating a forwarding path between the publisher and the subscriber. The path is encoded into a forwarding identifier (FID). Once the FID is computed, the topology manager forwards the signal from rendezvous node and the particular FID to the publisher. This triggers the

publisher to start the data delivery, in accordance with the topology manager's computation, by inserting this FID into the packet's header.

The entities that forward packets in the pub/sub network are *forwarding nodes*. When the forwarding node gets a packet, it parses the FID in the header, and chooses an outgoing link accordingly.

The publisher publishes the data based on the subscriptions. When the subscriber is no longer interested in receiving the data, it unsubscribes the information. The rendezvous system, again updates the information tree by excluding this subscriber from the publication. The topology manager calculates a new FID, and sent it to the publisher so that the data is forwarded only to the active subscribers. If there is no subscriber left for this publication, the publisher is signaled to stop publishing the data.

2.1.1.1 Information Structure

Information structure is a core element of the PURSUIT communication model. It is a hierarchical tree, consisting of two types of information: *Scopes* and *Information Items*. Every data transfer actively takes place by making a reference to this tree.

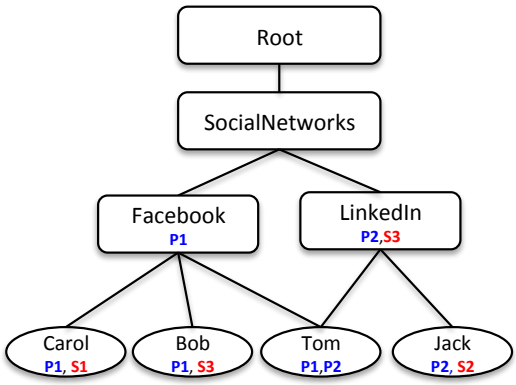


Figure 2.3: Example of the information structure

Figure 2.3 is illustrating a simple information structure. The letter *P* stands for a publisher and *S* stands for a subscriber. The rectangular boxes in the figure represent the scopes, whereas the oval ones are the information items. Each scope and information item is identified by its own identifier (ID), called *Scope Identifier* (SID) and *Information Identifier* (RID)[1] respectively.

The scope defines an area or a set of information, grouping similar information items together. The root scope is always at the uppermost level of the information tree. The scope can be declared under another scope, as we can see from the example. This introduces levels of information. According to an example in Figure 2.3, the information scoping allows the subscriber to subscribe the SocialNetworks as a whole or to choose only Facebook [15] or LinkedIn [33].

Information item represents the actual piece of data. It is placed under one or multiple scopes. More importantly, the information ID of each information item has to be uniquely assigned within the same scope, leading to the global uniqueness of an information item.

[1] The abbreviation RID comes from Rendezvous Identifier, which was previously used in the PSIRP project

2.1.1.2 Communication Strategies

PURSUIT architecture describes three basic types of communication strategies:

- **Node-local strategy**. Different processes, within a single node, communicate with each other. The information structure and topology graph are locally maintained in the node.
- **Link-local and broadcast strategy**. The information is available only for the local node and its closest neighbors, connected by a broadcast link. The subscriber subscribes to the information item. Then, data from the publisher can be either forwarded to the subscriber on specific links, or broadcasted.
- **Domain-local strategy**. This strategy covers the communication between all network nodes within a domain. In this case, at least one node in the network is chosen to be the rendezvous node. Also, at least one node takes the role of topology management. The topology graph is computed according to active nodes and links in the network. Then, the topology manager uses the graph to calculate an optimal path between the publisher and subscriber.

2.1.1.3 Service Model

According to the PURSUIT's concepts [13], there are basically six types of services provided in the Blackadder prototype.

Publishing and Unpublishing Scopes are ones of the essential services. The prototype allows user to declare a new scope to the network via a *publish_scope(id, prefix, strategy)* function. The *id* represents the scope identifier itself, while the *prefix* is the IDs of higher scopes, starting from the root scope. NULL value or empty string stands for the root scope in this case. The *strategy* is the communication strategies, discussed above. In this thesis we use the Domain-local (DL) strategy, since we would like to enable the communication between the network nodes. Moreover, the prefix needs to be the IDs of an existing scope. Otherwise, the network will reject our request. If we want to declare the scopes in Figure 2.3, the order will be.

1) publish_scope("SocialNetworks", NULL, DL)
2) publish_scope("Facebook", "SocialNetworks", DL)
3) publish_scope("LinkedIn", "SocialNetworks" DL)

In order to unpublish the scope, the publisher has to execute an *unpublish_scope(id, prefix, strategy)* request. The rendezvous system will delete the metadata of this scope, and also other scopes and information items under it.

Publishing and Unpublishing Information Items are used to announce the metadata of the actual information piece. These commands are issued by the publisher, through *publish_info(id, prefix, strategy)* and *unpublish_info(id, prefix, strategy)*. Again, taking Figure 2.3 as an example, the order of publications will be as follows

1) publish_info("Carol", "SocialNetworks" + "Facebook", DL)
2) publish_info("Bob", "SocialNetworks" + "Facebook", DL)
3) publish_info("Tom", "SocialNetworks" + "Facebook", DL)
4) publish_info("Jack", "SocialNetworks" + "LinkedIn", DL)
5) publish_info("SocialNetworks"+"Facebook"+"Tom",
   "SocialNetworks" + "LinkedIn", DL)

In step 5, Tom's information is declared differently. It is because *Tom* RID is a subset of both *Facebook* and *LinkedIn*. Also, it has been placed under *Facebook* already in step 3. Therefore, to re-declare an existing item, we insert the full ID of Tom as an *id* parameter, followed by the new prefix in the same form as we usually do. With this format, the system makes a connection between Tom's IDs; otherwise, we will see two separate *Tom* RIDs under the *Facebook* and *LinkedIn* scopes instead.

The *unpublish_info(id, prefix, strategy)* is used to inform the network that the publisher will not provide any data related to this *id* anymore. The rendezvous will remove this publisher from the information item. If there is another publisher, the RV will assign a new publisher for the corresponding item, and signals it to start publishing the data. Otherwise, that information item will be deleted from the information tree.

Subscribing and Unsubscribing scopes are supported by *subscribe_scope(id, prefix, strategy)* and *unsubscribe_scope(id, prefix, strategy)* respectively. For the *subscribe_scope*, the rendezvous updates publisher and subscriber pairs of all information items under that scope of interest, and signals the publishers to deliver relevant data to the respective subscriber. As a result, the subscriber will get the data updates from every item under that scope as well as the event, notifying the subscriber when a scope underneath is published or deleted. For example, if one pub/sub entity subscribes to the *SocialNetworks* scope before any other items is published, that entity will get a notification when Publisher1 and Publisher2 publish *Facebook* and *LinkedIn* scopes respectively.

Each subscriber is disassociated from the scope right after it sends an *unsubscribe_scope* request. The subscriber will stop getting any information related to that scope. Finally, the scope will be deleted from the database, if there is no existing publisher or subscriber at that scope, and also at any other subscope or information item under it.

Subscribing and Unsubscribing information items, with *subscribe_info(id, prefix, strategy)* and *unsubscribe_info(id, prefix, strategy),* the subscriber can address or withdraw its interest to/from an individual information item. The rendezvous repeats the matching and signaling processes so that the data transfer is started or stopped, when it is required.

Publishing Data is possible only at the publisher side. The publisher uses *publish_data(id, strategy, [data])* every time it has to publish the actual data, such as Carol's Facebook page, or its updated version. The *id* stands for a complete identifier, from the root scope to the information's RID. Therefore, if Publisher1 in Figure 2.1 wants to publish Bob's Facebook page, it will call the function below.

publish_data("SocialNetworks" + "Facebook" + "Bob", DL, [Bob's Facebook part 1])

publish_data("SocialNetworks" + "Facebook" + "Bob", DL, [Bob's Facebook part 2]) …

In general, the whole content of information is bigger than the packet size. Therefore, the data needs to be fragmented and sent in a form of data stream. The *publish_data* function is called when each fragment is delivered. Finally, the application on the subscriber side will take care of assembling them.

From this example, we can also see that Publisher1 does not have to indicate Subscriber3 as the destination. It just has to publish data under the topic defined, while the network takes care of all packet delivery.

Detecting pub/sub notifications is the last type of services. Publisher and subscriber call a *getEvent()* function to detect an upcoming signal. The events' notifications in PURSUIT

architecture are summarized in Table 2.1. Each notification indicates the type of event, related topic-identifier, and necessary FID for packet forwarding.

Table 2.1: PURSUIT publish/subscribe events

Event Types	From	To	Meaning
START_PUBLISH	RV/TM	Publisher	Start publish data, related to the topic ID, using the FID encapsulated with the signal
STOP_PUBLISH	RV/TM	Publisher	Stop publishing the data, under the topic ID
SCOPE_PUBLISHED	RV/TM	Subscriber	New scope is published under the father scope.
SCOPE_UNPUBLISHED	RV/TM	Subscriber	The scope is unpublished from the father scope.
PUBLISH_DATA	Publisher	Subscriber	Data under the topic ID has arrived

2.1.2 Rendezvous System

The rendezvous system performs information-related services according to the user requests [14]. It maintains the information tree by inserting or deleting information IDs, stated in the publish or unpublish requests. Also, it matches scope or data subscriptions with the publication in order to identify a node ID of the potential publisher and the subscriber's.

After the RV has gathered all relevant NIDs, it identifies a signal specific type, the signal destination, and the information identifier within the network. Later on, it cooperates with the topology manager to forward this signal to the relevant entity.

If the first subscriber subscribes to an available item, the rendezvous will immediately create a START_PUBLISH event towards the publisher. On the contrary, if the publication does not have any subscriber left, the rendezvous system creates a STOP_PUBLISH event, resulting the publisher to stop forwarding the data. With these events, the data is published only when needed, providing an optimal usage of the link capacity.

In addition, there are two other possible events, which take place after the subscriber subscribes a scope. These events are pointed towards the subscriber, notifying the subscriber about what is happening under the scope of interest. When a new lower-layer scope is published, the subscriber is notified by a SCOPE_PUBLISHED event. The subscriber can make a decision to further subscribe this subscope or just ignore the notification. On the other hand, a SCOPE_UNPUBLISHED event is used to inform the subscriber that one subscope is excluded from the tree. Thus, the subscriber will not get the data from this scope anymore.

2.1.3 Topology Management

The topology management is a system, controlling all packet deliveries within the publish/subscribe network. We call a node with this functionality a "topology manager". The topology manager stores link IDs and connectivity information in the form of a network graph. It has knowledge of the whole network, takes a role to compute the optimal path between the publisher and the subscriber(s), and create the forwarding identifier (FID) between the network entities, accordingly.

Once the topology manager has received an event from the rendezvous system, e.g., START_PUBLISH, it calculates two forwarding IDs: One is to forward the signal from itself to the publisher or the subscriber. The other one is the calculated FID from the publisher to the subscriber; this FID is encapsulated in the signaling packet. Finally, the topology manager uses the former FID to forward this signal to the corresponding entity.

As soon as the publisher gets the START_PUBLISH event, it starts the data publication by using the encapsulated FID as the forwarding information. The topology management keeps updating the FID as necessary; for example, when there is a new subscriber or when the subscriber unsubscribes the information.

2.1.4 Bloom Filter-based Forwarding

The PURSUIT architecture utilizes Bloom filter [6] based forwarding, described in the Line Speed Publish/Subscribe Inter-Networking (LIPSIN) [44]. Instead of identifying a node with an IP address, the pub/sub links are identified by link identifiers (LID). The topology manager uses LIDs to compute an FID, which is encapsulated in the packet header. The forwarder will read the header, and then uses it as a criterion to choose a set of outgoing links. This can be referred as a source routing mechanism, since the publisher specifies forwarding information before sending the packet.

Conceptually, each link is assigned by two LIDs: one for incoming and the other for outgoing direction. The link ID is an `m` bits binary-name, with a large `m` and a small amount of bits set to 1, e.g., a 248 bits identifier with 5 bits set to 1 and the remaining bits set to 0.

The topology manager constructs an FID by ORing the LIDs along the forwarding path. This FID is encapsulated in the signal to the publisher, where the publisher will use in a packet header of the PUBLISH_DATA event. This packetized Bloom filter-based FID is called a "zFilter".

Figure 2.4 shows an example of Bob's data forwarding, from Publisher1 to Subscriber3. As we can see, the zFilter is used as an outermost field in the packet header. Forwarding nodes analyse the header by ANDing the zFilter with all its adjacent LID(s), one at a time. If the result is exactly the same as the LID, it means that the corresponding link matches. Hence, that link will be chosen as the packet's egress link. This explains why FW1 and FW2 choose their eth2 as the outgoing interfaces.

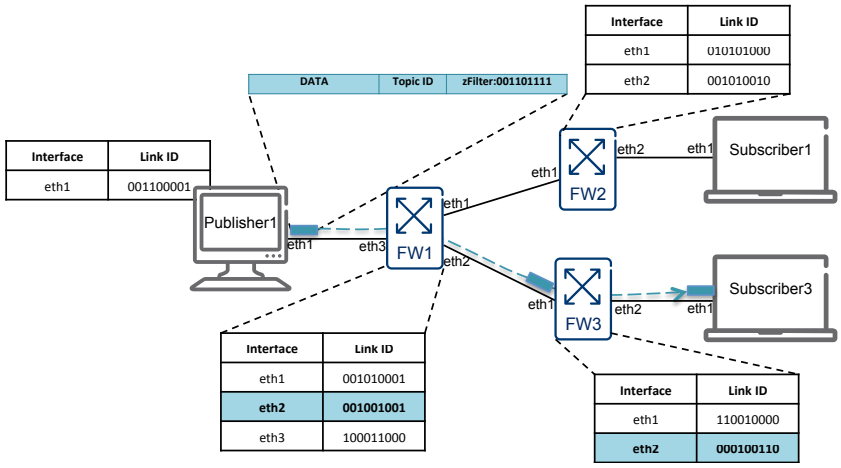


Figure 2.4: Example of in-packet Bloom filter forwarding

The zFilter provides multicast support by nature. The complete multicast tree can be encoded in the zFilter, the same way as described. For a case where the packet needs to be forwarded to several interfaces, the ANDing result will give more than one matching link IDs. This way, the forwarding node conveys the packet to all of those links. We call this a stateless multicast, since no group-specific state is required.

After the zFilter was introduced some researches for improvements have been proposed. Scalability issue is enhanced by the Bloom filter-based Relay Architecture (BRA), suggested by P. Jokela in [13] (page. 26-28). In addition, the anomalies, e.g., packet storms, forwarding loops, and flow duplication, are explored. The proposals to prevent them are presented in [40]. Also, Denial-of-Service protection using in-packet Bloom filters is suggested in [8].

2.2 Multicast Techniques

Multicast is a technique, providing data delivery from one or more senders to multiple receivers. Instead of sending duplicated copies to all recipients, the source of information in multicast network sends the data only once. Then, intermediate nodes take care of the packet forwarding and multiplication until it reaches the all destinations.

If we compare the multicast technique to broadcasting, the packets are transmitted to only interested receivers. On the contrary, the broadcasting uses additional bandwidth in delivering packets to uninterested receivers as well. For these reasons, the multicast is considered as an efficient method, which helps reducing the requirements of high bandwidth and broadcast channel-coverage.

In this section, we explain multicast technique in the IP network, followed by a comparison between the IP and PURSUIT zFilter multicasts.

2.2.1 IP Multicast

IP multicast technique consists of three basic steps. Fist of all, the sender sends packets to a multicast address. Secondly, the routers keep the states of the multicast group. Finally, the receivers have to indicate their interest to receive the packet from this multicast address by joining the groups, which triggers the routers to start conveying traffic from each specific group to them.

When IP version 4 was divided into four classes, from A to D, multicast addresses were in Class D [52]. They are confined within the range from 224.0.0.0 to 239.255.255.255 with the first four bits being always 1110. Furthermore, this class of addresses is separated into several scopes for more specific use cases.

The multicast session cannot be established until the receiver expresses its intention to join the multicast group. Therefore, Internet Group Management Protocol (IGMP) [5] is used for this purpose. It updates the group's membership when the receiver joins and leaves the session. This causes the routers to modify their own routing table as well. The methods of how the routers build up the path and forward multicast streams from the senders to interested receivers are called *Multicast Routing* [47].

2.2.1.1 Multicast Address and Scoping

The address scoping in IP multicast is used to regulate the traffic so that it stays inside one specific network area. In addition, these addresses also refer to the protocol scope, where we can find the packet type we would like to acquire. For instance, if we want to receive global SAP announcements, we should join the session at 224.2.127.254.

There are mainly 3 multicast address spaces defined by IANA [22]. Table 2.2 summarizes address spaces in accordance with RFC 5771 [34] and RFC 2365 [12].

Table 2.2: Multicast Address Spaces defined in RFC 5771 and RFC 2365

Address Range	**Designation**
Link-local Scope	
224.0.0.0 - 224.0.0.255	Local Network Control Block
Global Scope	
224.0.1.0 - 224.0.1.255	Inter Network Control Block
224.0.2.0 - 224.0.255.255	AD-HOC Block I
224.1.0.0 – 224.1.255.255	ST Multicast Groups
224.2.0.0 - 224.2.127.253	Multimedia Conference Calls
224.2.127.254	SAPv1 Announcement
224.2.127.255	SAPv0 Announcement
224.2.128.0 - 224.2.255.255	SAP Dynamic Assignment
224.3.0.0 - 224.4.255.255	AD-HOC Block II
224.252.0.0 – 224.255.255.255	DIS Transient Groups
232.0.0.0 - 232.255.255.255	Source-Specific Multicast Block
233.0.0.0 - 233.251.255.255	GLOP Block
233.252.0.0 - 233.255.255.255	AD-HOC Block III
Administrative Scope	
239.0.0.0 - 239.255.255.255	Administratively Scoped Block

2.2.1.2 Joining and Leaving Multicast Group

IP hosts and routers join and leave the multicast group by sending an IGMP report. In this thesis, the IP address for a group membership report is 224.0.0.22, since we focus only on the IGMP version 3 [5]. Message format of the IGMPv3 is shown in Figure 2.5.

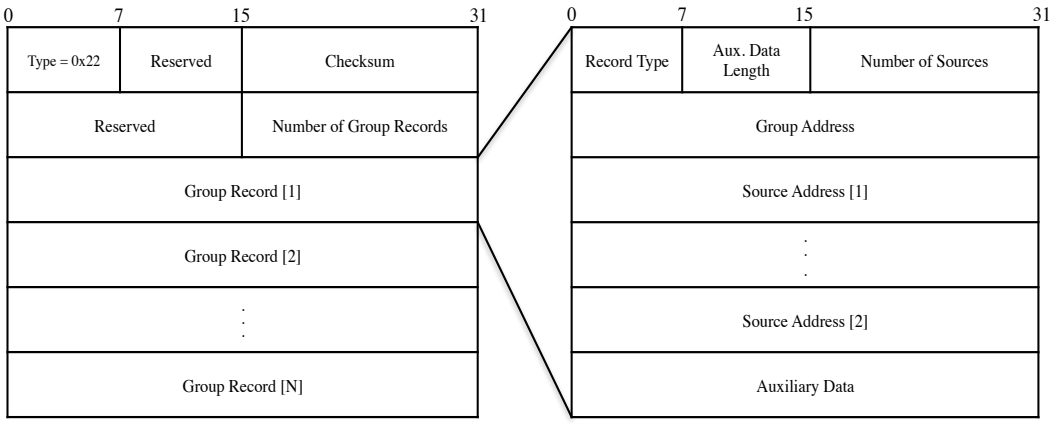


Figure 2.5: Packet Format of the IGMP Report

The format in Figure 2.5 is a common form for every IGMP message. We can differentiate the record type by looking at the *Record Type* field, which is the first byte of each *Group Record*. When an application joins the multicast address, it sends an IGMP message with CHANGE_TO_EXCLUDE_MODE (0x04) record type. The receiver expresses the multicast group it wants to join in the *Group Address* field. On the contrary, the record type

changes to CHANGE_TO_INCLUDE_MODE (0x03), when the application wants to leave the multicast group.

2.2.2 Comparison between IP and Bloom Filter-based Multicasts

States

From Section 2.2.1, we could see that the IP multicast is stateful, while the zFilter multicast is stateless. The IP router has to store and maintain multicast related addresses in Table 2.2, as well as the routing table for multicast forwarding. Meanwhile, it is claimed in [44] that zFilter has an advantage over the IP multicast, since the forwarder has a very small likeliness to create any forwarding state. In other words, there needs to be a large number of multicast groups or number of subscriptions in the network, before additional state is necessary.

Group Management

In the publish/subscribe network, the rendezvous system takes care of group management, instead of the routers in the IP. As a result, the forwarder is kept simple, and does not have to be aware of the multicast group forwarding. Moreover, IP router has to detect and process the IGMP requests locally.

2.3 Domain Name System (DNS)

Domain Name System (DNS) is a hierarchically distributed scheme for naming computers, services, domain, or any kind of the Internet compartments [46]. It generalizes the schematic and principle of the Internet applications, as the applications have become more sophisticated [45]. This section introduces the basic concept of DNS address resolving, and its message format.

2.3.1 DNS Basic Configuration

The basic configuration of the Domain Name System is shown in Figure 2.6. The user program resolves an IP address of each domain by firstly sends a DNS query to the resolver, which is a local host's interface to the domain name server. The resolver relays the request to the DNS server, which after that searches for matching addresses in the database. Finally, the server responds with a complete answer, an error, or a reference to some other DNS servers.

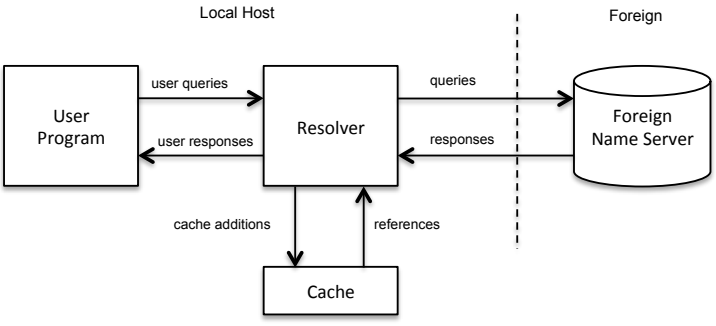


Figure 2.6: DNS basic configuration

2.3.2 DNS Messages

In this section we look at DNS message format, and show how DNS query and response look like. The DNS query and response share the same message format. The message has five sections: Header, Question, Answer, Authority and Addition. Normally, DNS query comprises of Header and Question fields, while a response is a combination between the previous two fields and at least one Answer.

Header is the uppermost field, where overall information is indicated. It specifies the type of message, whether it is a query or a response. Also, it tells the number of following fields, which come along with the header itself.

Question is the first field after the *Header*. The format of DNS question is shown in Figure 2.7. QNAME is the domain name we would like to inquire. It can be any length, but the name has to be terminated with a zero length octet before the type of question, or QTYPE, is appended. Lastly, a two octets QCLASS, expressing the class of a DNS query, is attached after the QTYPE.

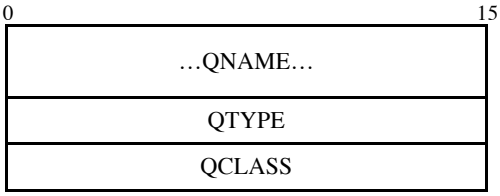


Figure 2.7: The format of a DNS Question

The *Answer*, *Authority*, and *Addition* fields are in the format presented by Figure 2.8, which is the same as a DNS Resource Records (RR). The information contained in the *Answer*, *Authority*, and *Addition* represents: answer to the query, the RRs pointing to an authoritative name server, and the additional query-related records respectively.

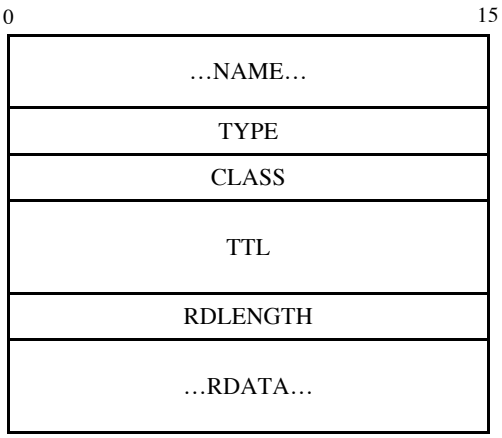


Figure 2.8: The format of DNS Answer, Authority and Addition

NAME is a pertained domain name. TYPE and CLASS are two octets values, characterizing the specific type and class of the domain. They are subsets of QTYPE and QCLASS in the *Question* field. QTYPE can be either a generic scope, such as "*****", to request for any kind of record, or it can be the TYPE value for more definite resolution. Different values of the TYPE field that we usually see in the network includes:

- A: An IP address version 4 of a host
- AAAA: An IP address version 6 of a host [54]
- CNAME: The canonical name for an alias
- MX: Mail Exchange
- SRV: The location of services [2]
- NAPTR: The naming authority pointer [39]

The same applies to the QCLASS. It can be "*", if the application makes a query for any class of record data (RDATA), or IN for the Internet, CS for the CSNET class, CH for the CHAOS class, and HS for Hesiod class. Of course, TTL, in the figure, is the time in second for the packet to be existed in the network, before being discarded. Finally, the message is appended with the length of RDATA, and the actual record.

2.4 Session Initiation Protocol (SIP)

The Session Initiation Protocol (SIP) [29] will be mainly used for testing the operation of the gateway. We have two major reasons to use SIP. First of all, SIP is a standardized application layer protocol with valid documentation. Secondly, SIP is utilized in multiple services, e.g. voice call, video call, conference call, and instant messaging. This section describes the basics of the SIP protocol as well as overall concept of the voice communication in this thesis. We use the SIP protocol for registering the users to our domain, determining their location and availability, initiating voice calls, and the conference calls between them. Finally, the SIP registrar is explained, since it is implemented to take care of the SIP domain in our pub/sub network.

2.4.1 SIP Overview

The Session Initiation Protocol (SIP) is designed for establishing, managing, and terminating multimedia sessions. When establishing the multimedia sessions, applications at end hosts need to have a common understanding of the media parameters. For example, the application must agree upon audio and video codecs as well as the session bit rate in order to get the correct playback. Therefore, SIP is used for negotiating the session's parameters and applications' functionalities to support SIP methods before the actual communication begins.

SIP parameter negotiation is based on an Offer/Answer Model [30]. The negotiation starts when one user sends an offer to the receiver. The offer contains sender's capability, i.e., its media preference parameters. The receiver can either reject or make a counter-proposal to the originator. If the offer is accepted, the receiver will send back the response message indicating acceptable media properties. Then, these two entities will have a common view of conceivable media parameters, and finally the media session is initiated.

Once the media session is established, SIP is responsible for session management and termination. This is because an ongoing session may need an adjustment; for instance, to initiate an additional video session or to start a conference call. When any of the users wish to terminate the session, SIP is used to inform the other session-holder about the termination. Ultimately, the other session-holder needs to acknowledge the session tear down, before the termination can be issued. These will be explained further in Section 2.4.4.

2.4.2 SIP Components

Any kinds of SIP sessions cannot be established without SIP components. According to [1], there are basically two types of the SIP components: SIP User Agents and SIP Servers. The SIP user agents are the representatives of physical devices, which communicate via SIP protocol. The SIP Servers, on the other hand, refer to logical applications, which sends a response according to the type of SIP requests. SIP user agents and servers can be divided into several smaller classes. They are introduced below.

2.4.2.1 SIP User Agents

In addition to being an end device communicating with each other, the user agent (UA) interacts with a user and takes action according to the user's input. The user can be human or other software. Every user agent is capable of originating, maintaining, adapting and terminating the session. It can transmit, retransmit, and discard the signal according to different circumstances, as well as understand the Session Description Protocol [38],

User Agent Client

A user agent client (UAC) is a device with a client application. It is a session originator, which sends at least six basic SIP requests [51]. Those are INVITE, ACK, OPTIONS, BYE, CANCEL and REGISTER.

User Agent Server

A destination of SIP requests is called user agent server (UAS). After the UAS receives a SIP request, it processes the request and sends either single or multiple responses back to the UAC. At this stage, the communication between the user agents begins.

Back-to-Back User Agents

Another possible type of the user agents is back-to-back user agent (B2BUA). It works as an intermediate entity, between the user agents. Instead of just receiving and relaying SIP request and response like a normal proxy, the B2BUA modifies some fields of the SIP message, and then sends it out as a new one. There are several reasons to use the back-to-back user agent. First of all, for security reason. The B2BUA hides the network's private information, e.g., IP addresses. Moreover, it provides call and internetworking management services. For example, call's billing, tracking, and adaptation.

SIP Gateways

SIP gateway is an application, connecting SIP-based networks with the ones using different protocols, enabling transparent communication between them. The gateway can get involved in only the signaling part, or both signaling and media sessions.

2.4.2.2 SIP Servers

As we have mentioned, the SIP server is an application, running inside the user agents. Therefore, there can be multiple SIP servers providing the services to one UA. The servers are categorized by their behaviour as stated below.

Proxy Servers

An application that responds to the SIP request, or forwards SIP messages to other entities is a proxy server. The proxy server is allowed to modify only some fields in the header, for a successful message routing, and preserving the transparency from the user agents' point of view.

A *stateful* proxy is a server that stores the state of the signal, keeps tracking, and perhaps issues some reactions due to the signaling state. For instance, the proxy might retain the state of a SIP request to make sure that the user gets the equivalent response. This allows the proxy to retransmit the same request, if the response is not received after some period of time.

In contrast, a *stateless* proxy routes the SIP PDUs just by using the information from the packet. It neither creates any states nor retransmits the packet.

Redirect Servers

A redirect server takes care of packet routing and responding like the proxy server; however, instead of relaying the packet like the proxy, it responds to the user application with *302 Moved Temporarily* message. In the *302 Moved Temporarily*, new information about the destination is stated, which instructs the UAC to reinitiate the session directly to the UAS.

Registration Server or SIP Registrar

Within an authoritative domain, a SIP registrar is allocated to take care of user registration. It receives a REGISTER message, and keeps the information in a form of device URI and Address of Record (AOR)[2] mapping. The place where this information is stored can be referred as a database or a location service. This registrar-generated database is also accessible by the domain's proxy and redirect servers, consulting them in packet routing. After this, another function of this registration server is to keep updating the users' status and their availability.

2.4.3 SIP Message Structure

SIP session management requires several pieces of information to be exchanged between the participants. The required information is arranged according to RFC 3261 [29], forming a SIP message.

SIP message consists of three parts, Start-Line, Header Fields, and Message Body. An arrangement of these components is shown in Figure 2.9. The first two fields are the Start-Line and the Header Fields, which are mandatory in the SIP signaling. The last field in SIP message is an optional one, called the Message Body. More detail of each field is described in the following subsections.

Start-Line
Header Fields
Message Body

Figure 2.9: The Structure of SIP Message

2.4.3.1 Start-Line

A SIP message can be either request or response message. The message type is denoted by the Start-Line. In case it is a request message, the Start-Line is called a "Request-Line". However, if the message is used for making a response to the respective request, it is called a "Status-Line". Both types of the Start-Line are structured as follows. SP and CRLF stand for single-space and end-of-line respectively.

[2] A SIP URI that points to a domain with a location service that can map the URI to another URI where the user might be available [29]

Request-Line

[Method] [SP] [Request-URI] [SP] [SIP-Version] [CRLF]

Figure 2.10: The format of SIP Request-Line

The first field in the Request-Line is *method*, an indication of the message purpose. In this thesis, we focus on the basic SIP methods specified in RFC 3261 as illustrated in Table 2.4. The next field is *Request-URI*, the destination's unique identifier. The last field is the *SIP-Version,* which is currently version 2 (SIP/2.0). The format and examples of the Request-URI are shown below.

sip:[username]@[domain name or IP address]:[port]

Figure 2.11: The format of SIP URI

Table 2.3: Examples of SIP Request-URI

sip:alice.blog@10.0.1.111 sip:alice@pubsubgateway.net:5060 sip:bobby.brown@pubsub.domain.com

Table 2.4: SIP methods from RFC 3261

Method	**Purpose**
INVITE	For session initiation
ACK	For acknowledging the session establishment or termination
BYE	For session termination
CANCEL	For canceling the previous request
REGISTER	To register username and location to the registrar
OPTIONS	For investigating other entity's capability

Status-Line

[SIP-Version] [SP] [Status-Code] [SP] [Reason-Phrase] [CRLF]

Figure 2.12: The structure of SIP Status-Line

The Status-Line starts with *SIP-Version*. Then, the version is followed by *Status-Code* and *Reason-Phrase*. These two parts specify the status of media session; for example, whether the call is accepted, rejected or pending. The Reason-Phrases relevant to the gateway implementation are presented in Table 2.5. SIP Reason-Phrases in Table 2.5 are the ones our gateway must pay attention to. Different meanings in Reason-Phrases help the gateway to manipulate the information structure in the pub/sub network.

Regarding the SIP responses, first of all the *200 OK* implies that the request is successfully fulfilled. It is also used to make a confirmation to the other entities that it agrees upon the connection status. The four latter Response-Phrases are for session termination. The callee sends *486 Busy Here* when he/she is unable to accept an additional call, whereas, the *603 Decline* is sent when the call is explicitly unwanted. In addition, *481 Transaction does not exist* and *487 Request Terminated* are meant for informing the other end host that the call is canceled after and before the callee takes action respectively.

Table 2.5: SIP Response-Phrases relevant to this thesis

Status-Code and Reason-Phrase	Meaning
200 OK	Successfully fulfill the request
404 Not Found	Destination of the call is unknown
481 Transaction does not exist	Request does not match the requirement
486 Busy Here	The callee is unable to accept this call
487 Request Terminated	The session is terminated by CANCEL or BYE method
603 Decline	The callee explicitly reject the call

2.4.3.2 Header Fields

Header Fields define more session-related information. The format of Header Fields is shown in Figure 2.13.

```
[header-name] [:] [SP] [header-value *(, header-value)]
```

Figure 2.13: The format of SIP Header Fields

In RFC 3261, there are 44 Header Fields. The names, meaning, and purpose of the most significant fields, used in the typical SIP applications, can be found in Table A.1, Appendix A. In this thesis we focus on the *Call-ID, Contact*, *From*, and *To* fields, which are the call specific identifier, the device URI, SIP URI of the call originator and the URI of the call destination.

2.4.3.3 Message Body

The last field, Message Body, is optional. It carries additional information about the media session. The Content-Type in the Header Fields is an indication of the Message Body type. For multimedia sessions, which are significant to this thesis, the Message Body is *application/SDP*, meaning that it will carry *Session Description Protocol (SDP)* parameters. The SDP carries information about the session owner, types of connection, supported media protocol and format, bitrate, and the format of specific parameters. SDP attributes, represented in required order, are shown in Table A.2. Each of the fields is representing different information necessary for session establishment and maintaining.

2.4.4 SIP Functionalities and Signaling

In each process, e.g., session negotiation and session termination, all the messages are exchanged through SIP signaling. In the thesis scenario, there are 3 major stages. The very beginning phase is SIP registration, where the user agent states its availability to accept the call. Then, the call is initiated, when a SIP user dials the call to the destination URI. Finally, the call is terminated. This section explains signaling exchanges in each stage.

2.4.4.1 SIP Registration

As it is shown in Figure 2.14, SIP registration involves two signaling messages: REGISTER and *200 OK*. It happens, when an account is activated. Obviously, the REGISTER is a message from the user agent to the SIP registrar. It requests the registrar to store a mapping between the user's public URI in the *To* header and the device URI from the *Contact,* in the database. Commonly, the public URI is "username@domainname", whereas the format of device URI is "username@ip_address", which indicates an exact routable location of the user agent. After the registration server finishes all the tasks, it sends *200 OK* back to the

particular end device, with the recently inserted AOR in the *Contact* field. This is to inform the user about a successful registration.

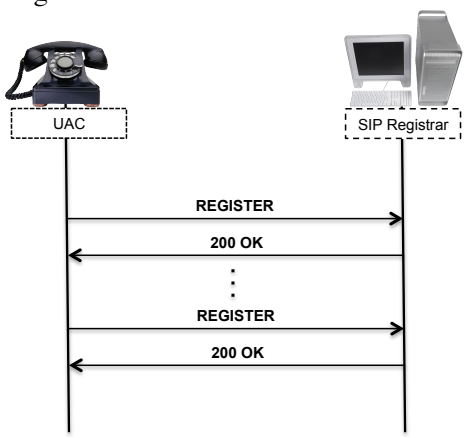


Figure 2.14: SIP registration example

The number in *Expire* header is an indicator of message registration type. It ranges from 0 to 4294967295. The number is 0 when the user logs out from the system, triggering the SIP registrar to delete the user entry. The process stops here without any SIP response. Apart from that, other positive numbers indicate the amount of time, the user would like to stay in each domain. In this case, the *Expire* will be included in the *200 OK* response, signifying the actual time allowed before the UA needs to send a new REGISTER message.

2.4.4.2 Call Initiation

SIP message exchanges in Figure 2.15 presents a simple session initiation procedure. A UAC sends an INVITE with the destination URI, its own URI, and the call's unique identifier in the *To*, *From* and *Call-ID* fields respectively. In addition to the header lines, the INVITE message comes with an SDP, containing all necessary information to establish the session as well as all possible media parameters that the UAC supports. Once the INVITE reaches the UAS, a *180 Ringing* response is sent back, telling the UAC that the destination UA has already received the request and the phone is ringing. In some situations, there is also a chance for *100 Trying* to occur before the *180 Ringing*, e.g., if the UAS is a stateful application.

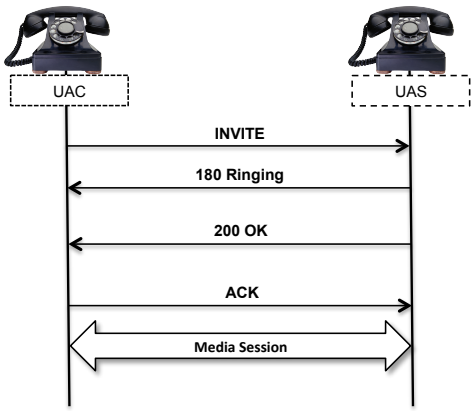


Figure 2.15: SIP call initiation example

As soon as the call is accepted, a *200 OK* with the UAS's media parameters is sent to the UAC. These parameters are the ones that both agents agree. The important media parameters for a voice call are, media format, port numbers, transport protocol, media codec, and sample rate. Then, the UAC acknowledge the UAS through an ACK message, before the media session is finally established between the two.

2.4.4.3 Call Termination

There are several reasons for the UA to terminate the call. We give three examples in Figure 2.16 and 2.17. The first one is when either participant hangs up the call. In this case, BYE is sent to the host on the opposite side. After that, the call termination is confirmed by *200 OK*. Here, both UAs terminate the session. If there are more than one ongoing sessions, the user agent uses the *Call-ID* as a reference.

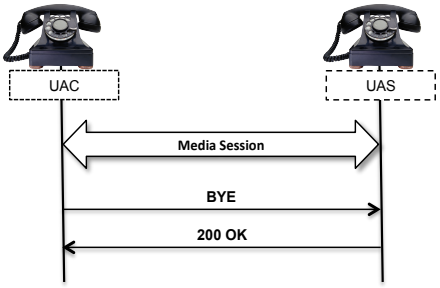


Figure 2.16: SIP call termination example

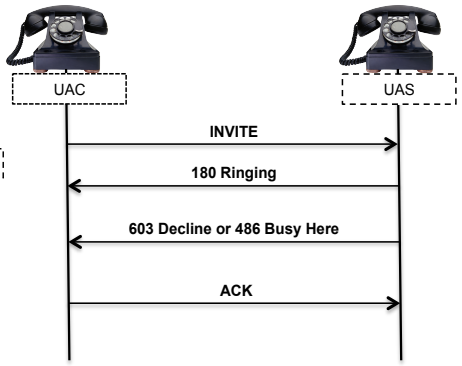


Figure 2.17: SIP call refusal example

Other two cases take place only at the callee side. At the time an application starts to ring, the callee can choose to reject the call by sending *603 Decline* or *486 Busy Here* back to the caller. Finally, the call is terminated after the callee receives an ACK.

2.4.4.4 Call Cancelation

The caller cancels the invitation by sending a CANCEL request in Figure 2.18 and 2.19. Figure 2.18 is illustrating the situation, when the UAS has not picked up the call before the request arrives. Therefore, the UAS just sends *487 Request Terminated*, and finally waits for an ACK from the UAC to finish the process.

The signaling is more complicated if the CANCEL is sent while the session is establishing. That means the client has already picked up the call before the CANCEL arrives. Hence, the UAC has to send an ACK, responding the *200 OK* and another BYE to terminate the call again. On the UAS side, after the application knows that the call is canceled, it respond to the acknowledgement by *481 Transaction does not exist*. Eventually, it sends *200 OK* again to assure the call termination from the BYE request.

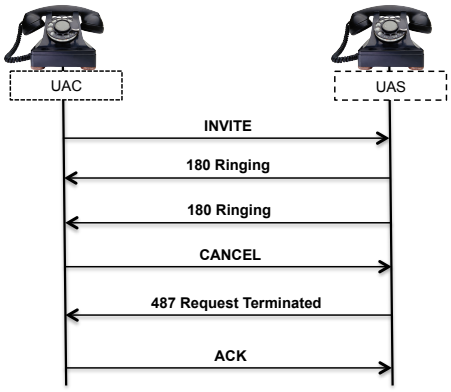


Figure 2.18: SIP call cancelation, when the caller gives up

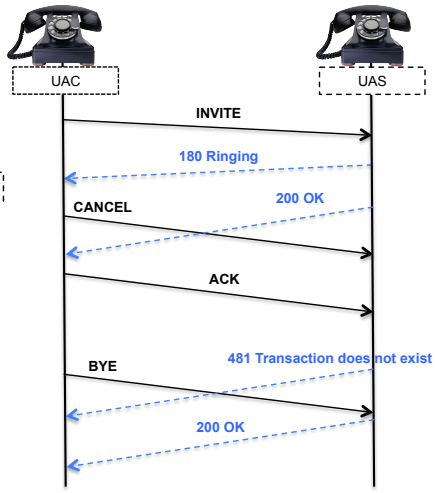


Figure 2.19: SIP call cancelation, while the session is establishing

2.4.4.5 SIP Conference

SIP conference is a real-time communication with multiple SIP participants. There are several scenarios for the SIP conference [31]. The first one is a loosely coupled conference, which utilizes the multicast technique. In this scenario, there is no explicit signaling between members of the conference group. The user persistence is learnt through some other control information, such as RTP Control Protocol [19] in a multimedia session.

Another model, where SIP messages are fully exchanged among all participants without any centralization, is called fully distributed multiparty conference.

The last one is a tightly coupled conference. It is a scenario where a centralized system is used as a point of connection. Unlike the previous cases, the interaction between participants is made through this central system only, both signaling and media. This thesis concentrates on this scenario, which is the most commonly used.

This central point of communication is called “focus”. It is a logical feature of a user agent, which understands event packages for conference states [32], maintains an ongoing conference, and makes sure that the media reaches every participant. Inside each focus, there is at least one mixer, combining media streams from all participants before it is sent out. In addition, it makes use of a few supplementary systems to help with the control mechanism. These are Conference Functions, and Floor Control Server with Binary Floor Control Protocol [17].

A user agent can also support SIP conference. It takes part in the conversation like the focus. This type of the user agent with call control functionality is called a “conference-aware UA” [3]. In this case, the signaling has to be centralized, but the media can be either multicast, distributed or centrally mixed. Examples of signaling and media centralization are presented in Figure 2.20 and 2.21.

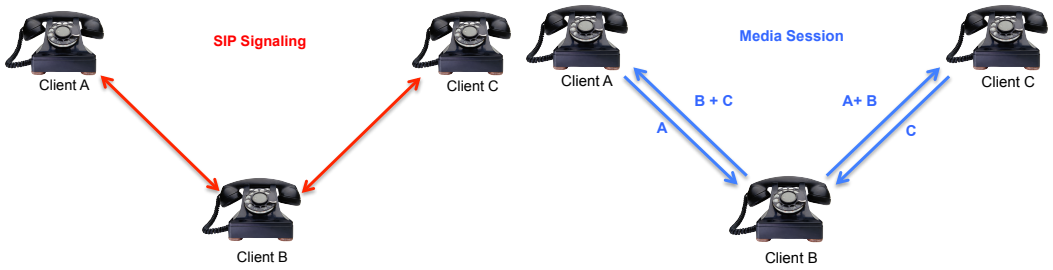


Figure 2.20: SIP signaling for tightly coupled conference **Figure 2.21:** Media session for tightly coupled conference

In brief, one of the simple ways to create a SIP conference is to separately send an INVITE to each participant. Then, the communication is centralized to a conference-aware UA. It mixes the media from other participants and forwards them, while being aware of the signals at the same time.

2.5 Session Announcement Protocol (SAP) for Multicast Session

Session Announcement Protocol (SAP) is a "multicast session directory announcement protocol" [37], for advertising information about existing multicast sessions. The purpose of SAP is to convey necessary parameters to the clients so that they could join the multicast session properly.

A SAP packet contains header and payload. The SAP header consists of SAP version, authentication length, message identifier hash, originating source IP address, and finally the payload type. A combination between the originating source IP and message identifier hash forms a unique identifier of each session. This helps the SAP announcer to identify all announcements within the network.

SAP messages are sent periodically; however, the time interval between each of them varies. The total number of session announcements is one of the factors, because the bandwidth for all announcements is predefined, for example, to be 4000 bits per second. Thus, the announcer has to monitor other announcements, based on their unique identification, to calculate the time difference between each of them.

Regarding the SAP destination, SAP message is sent to port 9875 of the well-known multicast addresses. They are 224.2.127.254 for global scope SAPv1 and 224.2.127.255 for SAPv0 [12]. Moreover, 224.0.0.255 is used to announce SAP messages for the link-local scope.

For RTP media streaming, the payload follows Session Description Protocol (SDP). The SDP is a part with required information for the client to join the multicast session, i.e., the session owner, multicast address, port, media description and media attributes. With the information in the SDP, the client can join the multicast streaming by indicating the multicast IP address and port to a media player. We can see an example in Figure 2.22. It demonstrates how IPv4 multicast works with the SAP protocol.

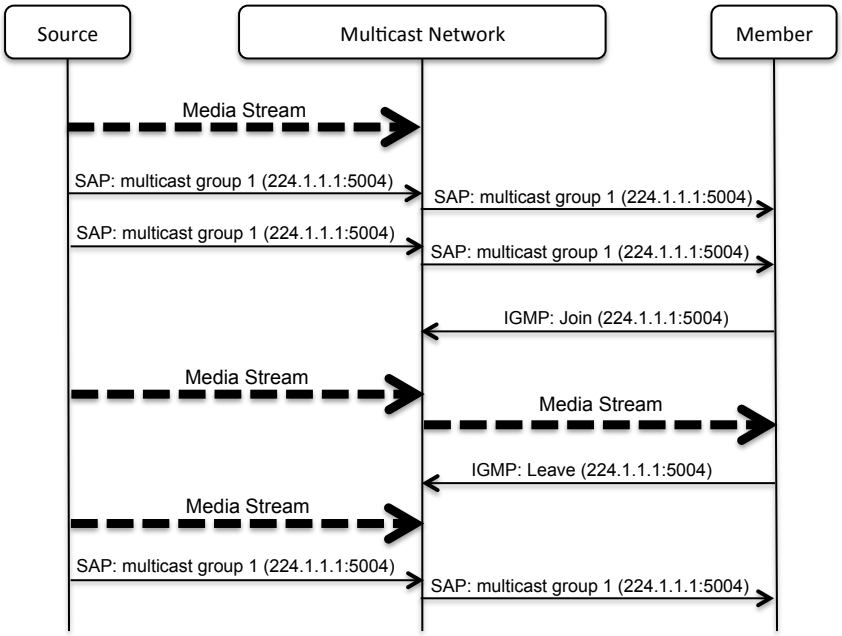


Figure 2.22: Example of IP multicast with SAP protocol

2.6 Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP)

Up until now, we have mentioned the possibilities to establish a multimedia session. In this section, we move on to consider how the multimedia, such as audio and video, are conveyed across the Internet. In this thesis, real-time[3] multimedia transfer [19] is handled by Real-time Transport Protocol (RTP) and RTP Control Protocol.

Conceptually, RTP and RTCP are application layer protocols, operating on top of two different UDP [27] ports. Usually, the RTP port is an even number, whereas, the RTCP port is the RTP's next higher odd port. For example, IANA [22] has defined official RTP and RTCP ports, which most applications use as an initial setup, to be 5004 and 5005. If there are multiple RTP and RTCP flows in one session, other dynamic ports [35] can be additionally allocated.

The media content can be transmitted with a constant frequency, such as every 20 ms, or its packetization interval can be altered according to an encoding mechanism as suggested by RFC 1890 [18]. Moreover, audio and video sessions can be transmitted in the same or separate RTP and RTCP ports. This facilitates the user to choose the media type to play out, i.e., only video, only audio, or both of them. Even though the audio and video are transmitted separately, they are synchronized at the receiver's end by using timing information from RTP, and NTP timestamp in RTCP reports.

RTP and RTCP work collaboratively with each other. Therefore, we briefly give an understanding of their packet compositions and their concepts in the next subsections.

[3] The system characteristic in which the sending and receiving rate are identical.

2.6.1 RTP

The RTP protocol is used to carry individual chunks of data. Therefore, each RTP packet is raw data encapsulated with an RTP header. The RTP header contains timing information for the media playback, along with the information of session participants. It also stores the payload type, telling about the media codec of the attaching data content. RFC 3551 [20] can be used as a reference, since it specifies default audio and video profile that could occur in the multimedia session. RTP header format is presented in Figure 2.23.

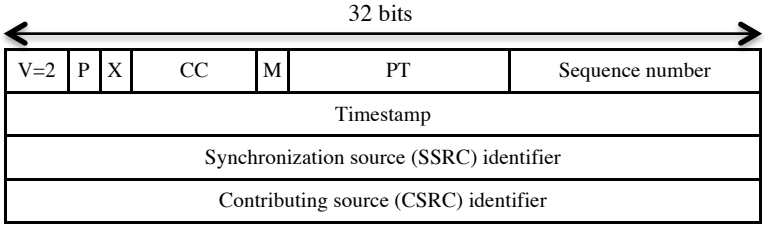


Figure 2.23: RTP header format

The sequence number and timestamp in the RTP header are the important timing information. When the sender starts to send the first packet, the sequence number is initially set to a random number. Then, it is increased by one in every new packet. This enables the receiver to detect the packet loss and restore the packet continuity. Next to the 16 bits sequence number, there is a 32 bits timestamp. It reports the time each packet is sent out, according to the media's sampling rate. Similar to the sequence number, the timestamp is randomly initialized; it is increased by the media's profile-specific sampling period, regardless of the actual time each packet is sent. For instance, if the sampling rate is 8000 samples per second, the increment will be 125 microseconds at a time. This feature is for jitter compensation and media synchronization.

Two participant-associated fields are Synchronization Source Identifier (SSRC) and Contributing Source Identifier (CSRC). The SSRC is an RTP session's unique number, which identifies each respective source of the media. Eventually, the standard RTP header ends with the CSRC, stating maximum 15 contributing source identifiers within the same session. These two fields help the receiver to differentiate the source of each packet, for example when the mixer mixes multiple media streams into one.

2.6.2 RTCP

In addition to RTP packets, the RTCP is periodically sent and received by all participants in the session. However, the RTCP carries only control messages. The protocol essentially provides: Quality of Service feedback reports, RTP source transport identifier or canonical name, and number of participants, which is used to calculate the intervals between each packet.

With the information from all RTCP messages, active senders and receivers can improve the quality of media stream. For instance, the message can be used to find the location of problem, whether it happens locally or globally. In addition, the participants can calculate network related parameter so that they can adapt, e.g., sampling rate or codec, accordingly.

We can find elaborated details of the RTCP messages in RFC 3550. This section presents generic details of them as follows.

2.6.2.1 Sender Report (SR)

Sender Report is a statistical report concerning packet transmission and reception. It is sent from an active sender to other participants within the same RTP session. The receiver can recognize the sender report by seeing *200* in the packet type (PT) field, and the packet is identified by the sender's SSRC. After the sender's SSRC, the sender report carries sender information block, holding NTP timestamp, RTP timestamp, the sender's packet count and the sender's octet count. Next fields are reception report blocks, and the sender report ends with profile-specific extensions.

Sender Information Block

The first information inside the sender information block is the NTP timestamp. It is the wallclock time[4], reported when the packet is sent. It is used with timestamps in the receiver reports to compute round trip time, between itself and the receivers. If the system does not know the wallclock time, NTP timestamp can also come from the system's common clock or just start with zero at the beginning of the session.

RTP timestamp presents time from the same referenced system as the NTP timestamp, but in the RTP units and the RTP random offset. Both sender and receiver use this timestamp to synchronize the media.

Regarding the amount of data, the sender's packet-count field gives an idea about the accumulated number of packets from the source, starting from the beginning of the session. Again, the report is identified by the specific SSRC. On the other hand, the sender's octet count is the cumulated amount of data itself, not including other headers. The sender's octet count is the last field in the sender's information block.

Reception Report Block

The reception report block comes after the sender information block. The number of attaching reception report block is corresponding to the number of other synchronization sources that the sender can detect. Each of the block starts with a uniquely assigned SSRC of the sender. The following fields give information about the packet lost, the highest sequence number received, an estimated variance of inter-arrival delay, the latest time the originating sender gets this specific source sender report, and finally the time interval between the latest received sender report and the time this report block is sent out.

2.6.2.2 Receiver Report (RR)

Similar to the sender report, receiver report gives information about the packet transmission and reception, but it is sent from the media's receiver instead. The message format of the receiver report is the same as the sender report, except that the value in packet type field is *201* and the first reception report block is appended to the receiver's SSRC without any sender's report in between.

---

[4] The report, telling the time used for one particular task to be finished. The time is started on 1 January 1900, 00:00:00 UTC, and continued until present.

2.6.2.3 Source Description Items (SDES)

The packet type with value *202* is an indication of a source description items. Inside the packet, there can be no data, or a number of chucks, containing additional information about each participant. Four octets of SSRC/CSRC appear first in each of the chunks. Then, it is followed by a list of additional information related to that SSRC/CSRC. Possible information can be; for example, a canonical name (CNAME), user name (NAME), e-mail (EMAIL), and geographical location (LOC).

2.6.2.4 BYE

The end of source's participation is notified by BYE report. It comes with value *203* in the packet type field. The BYE packet contains SSRC of the leaving source. Optionally, the packet can state a reason for the source to leave after the SSRC. If a mixer is shut down, it will list the CSRC of all contributing sources, connected to itself, in the BYE packet before the BYE is sent out.

2.6.2.5 APP

Application-specific function, with *204* packet-type identifier, is reserved to be used by new applications and features. The RTCP header fields are kept to be the same, followed by the ASCII name defined by an application and the application-related data.

2.7 Summary

The Session Initiation Protocol enables users to register to a public domain, initiate, modify, and terminate the voice communications. These procedures are accomplished by several exchanges of the SIP requests and responses. After the session is successfully established, the voice packet between the SIP entities are conveyed by the Real-time Transport Protocol and control messages are transferred by the RTP Control protocol.

Similarly, the RTP and RTCP protocols are used to transfer multimedia content. The information about the multimedia session is announced by the Session Announcement Protocol, which is then joined and left by the end users through the Internet Group Management Protocol.

Instead of the normal send/receive model in the current Internet architecture, our thesis utilizes the PURSUIT's publish/subscribe network to connect between the IP hosts. In the next chapter, we discuss what we would like to achieve from the design of our pub/sub gateway and SIP registrar, within the scope of voice communication over SIP and multimedia streaming over the multicast.

Chapter 3

Requirements

In this chapter we discuss the requirements of our pub/sub gateway and SIP registrar in accordance with the general goal of this thesis, defined in Chapter 1. We start from relevant circumstances, in SIP and multicast media streaming, which our design needs to support. They cover session establishment, session termination and also other possible situations that might occur during an ongoing session.

The requirements of both the pub/sub gateway and SIP registrar are defined so that they enable the communication between IP hosts over a network, based on the publish/subscribe paradigm. They are separated into two parts. First, the operational requirements of our gateway and SIP registrar are summarized, followed by other important aspects for the communication scheme.

3.1 Network Overview

Figure 3.1 illustrates an example of our network layout. The gateways will be installed at the edge of the publish/subscribe network, acting like an ingress/egress point of IP traffic to and from the pub/sub network. In this scenario, the IP traffic is generated by Alice, Bob, Carol, Dave, and Eve. SIP registrar is placed inside the pub/sub network, dealing with user registration, based on the concept in Section 2.4. In the following subsections, we give an overview of SIP voice call and multimedia streaming over multicast, followed by the requirements of our design based on the figure below.

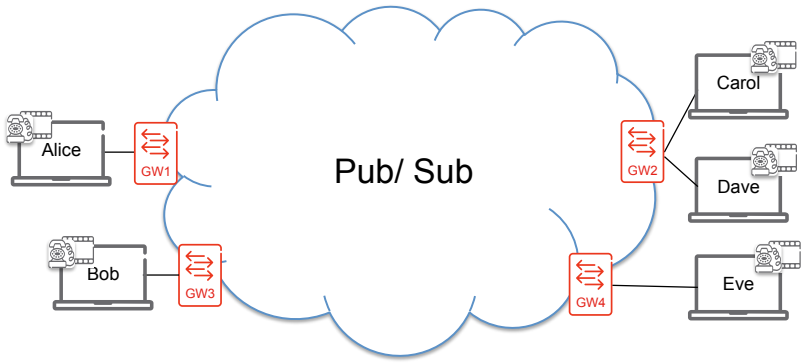


Figure 3.1: Network overview

3.1.1 SIP Voice Call

In order to set up a SIP call from Alice to Dave, SIP applications need to register Alice's and Dave's username to the domain. Next, Alice enters Dave's URI and starts the call. If Dave is online and ready to accept the incoming call, his SIP application starts to ring. Otherwise, the call is dropped. While the application is ringing, Dave can choose whether to accept or reject the call. If the call is accepted, a media session is initiated between Alice and Dave using RTP and RTCP protocols [19]. The session will be maintained until either of them terminates it.

During the ongoing SIP session, it is possible that Alice or Dave will temporarily pause the call. This means they will not be able to hear each other's media. In addition to holding the call, SIP users can make a conference call between more than two SIP user-agents in a group, as well. For example, Alice might invite Carol and Eve to join the conversation.

3.1.2 Multimedia Streaming over Multicast

In multimedia streaming, we simulate the situation when Alice streams a video file to an available multicast address with Session Announcement Protocol [37]. The SAP message is periodically announced to the address 224.2.127.254 and port 9875 as mentioned in Section 2.5. This SAP message contains all relevant information that the user need for subscribing each specific media stream, such as address and port. Bob, Carol, Dave and Eve can examine the SAP announcement, and join the multicast session using a media player. A user who has already subscribed to this multicast group can also leave the session by sending a signal to this multicast address. The signals for session joining and leaving are handled by the IGMP CHANGE_TO_EXCLUDE_MODE and CHANGE_TO_INCLUDE_MODE respectively [5].

3.2 Functional Requirements

In this section, we consider requirements of our system as well as how the pub/sub information should be manipulated to support SIP and multimedia streaming.

3.2.1 SIP Voice Call

3.2.1.1 User Registration and Registration Withdrawal

SIP user-registration is a process, where the application sends a REGISTER message to the network. Our gateway needs to convey this REGISTER message to the SIP registrar, located in the pub/sub network. In addition, the gateway has to correctly route a response from the registrar back to the corresponding host as well.

SIP registrar is a very significant contributor in this process. One of its requirements is to store information, required for call forwarding in both pub/sub and IP networks. Besides, it should be able to delete the user entries according to a deregistration request and timeout period.

3.2.1.2 Call Signaling

When Alice sends an INVITE to Dave, Gateway1 should cooperate with the SIP registrar so that it can accurately forward the INVITE request to Gateway2 on Dave's side. At the same time, Gateway2 has to forward this INVITE and other SIP messages to the right host inside its IP administrative area. Finally, these gateways need to maintain the status of this session

in accordance with SIP message exchanges between Alice and Dave, i.e., session teardown, call cancelation, and rejection.

3.2.1.3 Media Session

After the call is established, a conversation between Alice and Dave starts. RTP and RTCP packets will be transmitted between the hosts. The gateway should be designed so that the media can be sent and received through the publish/subscribe network without any package drop. Moreover, if there are multiple media sessions at the same time, these sessions should be handled independently.

3.2.1.4 Support for Multiple Call States

Apart from an active call, this thesis takes into account the conference call and the call holding as well. Therefore, our gateway has to support these features over the publish/subscribe network as well.

3.2.2 Multimedia Streaming

3.2.2.1 SAP Announcement

Any of the IP hosts acts as a multimedia server by transmitting media content to a multicast address. In our sample case, we chose Alice to be a server. Also, we assume that SAP announcement is enabled together with the media streaming. Therefore, the gateway has to keep listening to the SAP announcement from Alice and forward the packets using the publish/subscribe paradigm. Meanwhile, the destination gateways, i.e., Gateway2 to 4, should be aware of SAP announcements from the pub/sub network, and convey the information to the SAP specific IP address.

3.2.2.2 Multicast Subscription and Subscription Cancelation

Bob, Carol, Dave, and Eve learn the information about Alice's media by inspecting the SAP announcement. Both multicast subscription and cancelation, from the clients, are indicated by the IGMP Protocol. Hence, one necessary requirement of our gateway is an ability to recognize IGMP packets, and distinguish their types.

The clients, other than Alice, join the multicast group by specifying Alice's multicast address and port in the IGMP "group join" message. The gateway has to parse this IGMP signal, and then precisely request the media from Alice. During the multimedia session, Alice's gateway has to convert RTP and RTCP packets into publish/subscribe traffic. Other destination gateways have to convert the media from pub/sub back to RTP and RTCP protocols, while streaming them to the IP multicast address at the same time. In addition to joining a multicast group, our gateway should handle IGMP "group leave" report and stop the media streaming, when necessary.

3.2.3 Publish/Subscribe Information Management

As we have described in Section 2.1, the communication within the pub/sub network is based on an information tree, where metadata and mappings between publications and subscriptions are maintained. Without proper information management, the previous requirements will not be successfully achieved.

To fulfil the previous requirements, SIP call and multimedia streaming should be independent from each other. Moreover, the information tree should be dynamically

arranged according to the call and multimedia status, so that only relevant states are kept in the network.

3.3 Other Requirements

Apart from the operations of our gateway and SIP registrar, the design should be appropriate for a more complicated network scenario. One such scenario is when multiple gateways have to work synchronously, either on the same or different sessions. Additional requirements for these aspects are defined below.

3.3.1 Performance

One intention of this thesis is not only to facilitate the Internet Protocol to communicate across the publish/subscribe network, but also to suggest an efficient design of the pub/sub gateway and SIP registrar. Thus, our design should enable the communication with a smallest effort, time consumption and minimal amount of states, both inside the network and the pub/sub elements themselves. Moreover, the gateway should know which protocol to utilize under different circumstances. For instance, it should choose the IP protocol to interact with the hosts inside the same subnet, while it should utilize the publish/subscribe paradigm only when needed.

3.3.2 Reachability

When there are two existing types of networks, interacting with each other, it is certain that IP users can be either in the same or in different IP subnets. The gateway should be designed so that SIP signaling and real-time multimedia can reach all the hosts regardless of their location.

3.3.3 Support for Concurrency

In this communication model, it is possible that the users will participate in multiple SIP or multicast sessions simultaneously. In this case, the gateway should maintain the previous ongoing sessions and concurrently support the recently generated one.

3.3.4 Adaptability

We assume that a host does not have any information about what is happening on the other side of the network, because the pub/sub gateways are in between. As a result, there might be some overlapping in address and port assignments of the SIP and multicast sessions. For example, when Carol replies to Bob's call with the media ports, which have already been occupied by Alice and Dave's conversation. Another example is when the clients from different IP networks, like Alice, Bob and Eve, stream the media through the same multicast address, e.g., with 224.1.1.1:5004. In these cases, the gateway should have a mechanism to detect the anomaly, and then adapt the session parameters so that the session is successfully established.

3.4 Summary

In this chapter, we have described expected results we would like to achieve. With our system design, one or multiple SIP voice calls should be possible between the IP hosts, both within the same and different IP subnets. Moreover, the gateway has to enable the hosts in joining/leaving the multicast session(s), and receive the multimedia content from any multicast group across the pub/sub network. The next chapter will explain our detailed design to accomplish the objectives described in this chapter.

Chapter 4

Design

This chapter presents the design of our IP to publish/subscribe gateway and SIP registrar. We start from an architectural set-up, showing how the gateway communicates with the IP clients and the pub/sub-based entities. After that, we describe the mechanisms of gateway and SIP registrar to support SIP registration, SIP voice call and multimedia streaming over the multicast technique.

4.1 Architectural Set-up

The architecture, depicted in Figure 4.1, is a basic set-up for the design in thesis. We design the gateway and SIP registrar based on the Blackadder prototype [23]. The prototype takes care of data publication, subscription, and packet forwarding. Our gateway and SIP registrar are system-level applications, operating on top of the Blackadder node.

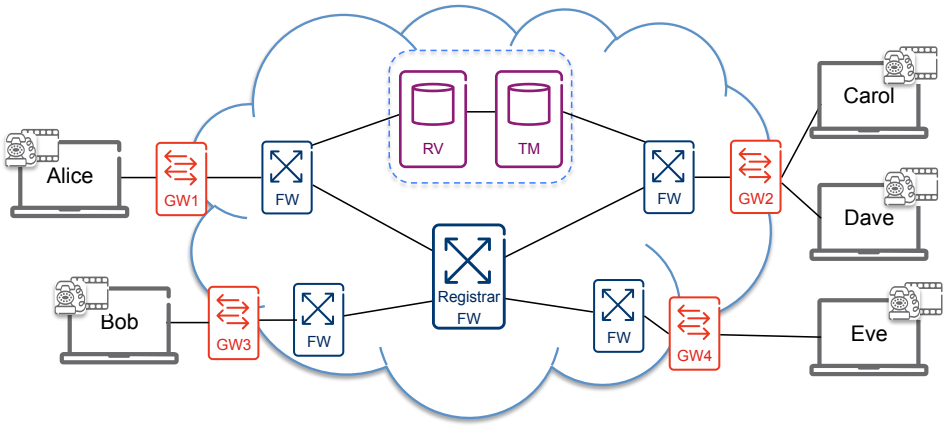


Figure 4.1: An architectural set-up for the thesis design

The gateway application operates at the intermediate node between IP and publish/subscribe networks. The SIP registrar is a pub/sub-only application. Our gateway uses the Internet Protocol to communicate with the clients, and uses the publish/subscribe paradigm to interact with other gateways and the SIP registrar.

We assume that the name of our SIP domain is "pubsubgateway.net", where all SIP users make a registration and indicate their availability to accept voice calls. In order to get related IP addresses of our pub/sub domain, SIP client has to send a query to a DNS server. Since there is no plan to install any DNS server in the pub/sub network at this stage, we assume that our pub/sub gateway also acts as a domain name server for each IP subnet. If the SIP client sends a query, asking for an IP address of the "pubsubgateway.net" domain, our gateway will respond with its own IP address. Otherwise, the DNS response will refer to another DNS server. In addition to the DNS support, we design the pub/sub gateway to be an intermediate application, converting IP to the publish/subscribe traffic and vice versa.

Our gateway executes data publication and subscription on behalf of every IP client in the attaching network. It is designed to apply information scoping, which classifies the packet into two basic sessions: ***SIP*** and ***Multicast*** sessions. In the next sections, we explain how the gateway maps pub/sub information under these two scopes, with the IP protocols.

The pub/sub-based SIP registrar follows fundamental requirements defined in RFC 3261 [29], except that it communicates through the publish/subscribe paradigm. It takes care of user management in the pub/sub domain by accepting a registration request, storing user's information in a location service, and publishing a response back to the corresponding gateway, indicating that the registration is successfully completed. Additionally, our SIP registrar accepts a call invitation from the user, and takes the responsibility to redirect that call to its destination. This includes relaying the call invitation to the destination gateway, and consults that gateway to route the call by the recipient's device address.

4.2 System Initialization

In the following sections, we discuss the behavior of our system at the time it starts to operate. The design and all figures will be described in correlation with the scenario in Figure 4.1. All abbreviations used in the state diagrams are listed in Table 4.1 below.

Table 4.1: Abbreviations in the state diagrams

Abbreviations	Meaning
PubSc	Publish Scope
PubIn	Publish Information
SubSc	Subscribe Scope
SubIn	Subscribe Information
Pub_Data	Publish Data
START_PUB	START_PUBLISH
STOP_PUB	STOP_PUBLISH
SCOPE_PUB	SCOPE_PUBLISHED

4.2.1 SIP Registrar Initialization

We assume that our SIP registrar is started before any gateways. Its pub/sub signals during system initialization phase are depicted in Figure 4.2. The registrar prepares the uppermost scopes for SIP sessions by publishing ***Register*** and ***Call*** scopes at the beginning. It has to be done because we separated the SIP session within the pub/sub network into two independent parts, which are user registration and the actual voice call. After publishing these scopes,

our registrar subscribes the ***Register*** scope to denote its participation in a user registration process.

In fact, any pub/sub entity can publish the top-level scopes for multimedia streaming, as long as it is activated before the gateway. In our design, we use the SIP registrar to publish ***Multicast***, and two underlying scopes named ***SAP*** and ***Media***. As the names imply, our gateway uses the ***SAP*** scope to publish multicast-session announcements, while the ***Media*** scope is to support the media content in each multicast session. The type of media content can be a combination between audio, video, text, or images.

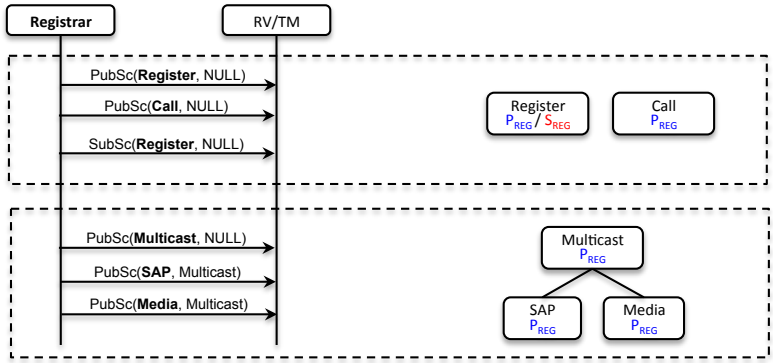


Figure 4.2: SIP registrar initialization

4.2.2 Gateway Initialization

Initiation processes of Gateway1 to 4, to support SIP voice call, are shown in Figure 4.3. When the pub/sub gateway starts running, it publishes its own unique identifier as a subscope under the ***Register***, previously published by the SIP registrar.

The gateway's ID under the ***Register*** scope represents an IP subnet, in which the gateway will make a publication and subscription for. It helps the SIP registrar to redirect a call from one IP network to another. Finally, the gateway has to publish an information item called *Signal* under its scope in order to announce its status, for example, when it is going to shut down.

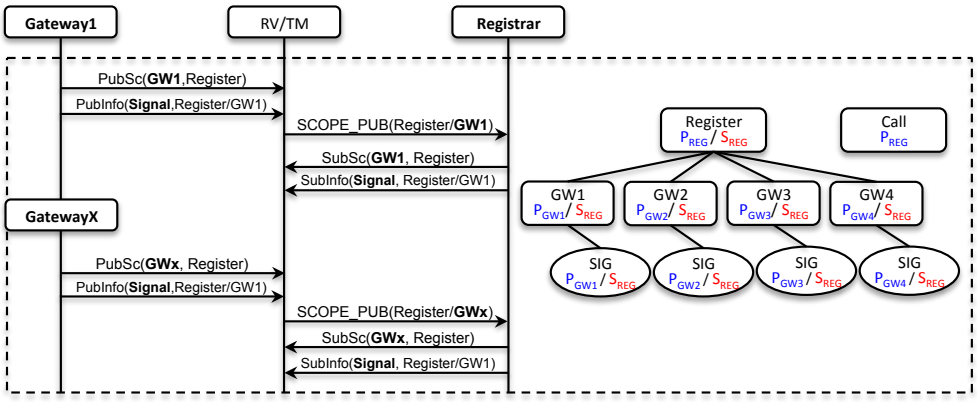


Figure 4.3: Gateway initialization for SIP support

The SIP registrar will get a SCOPE_PUBLISH event, notifying about the gateway's publications. It causes the registrar to make a subscription to the gateway's scope and *Signal*

RID, for stating its participation in user registration processes as well as its interest to know the status of each gateway, respectively. At this stage, the gateways are ready to process any kind of SIP request from the IP clients, and the REGISTER request will reach the SIP registrar.

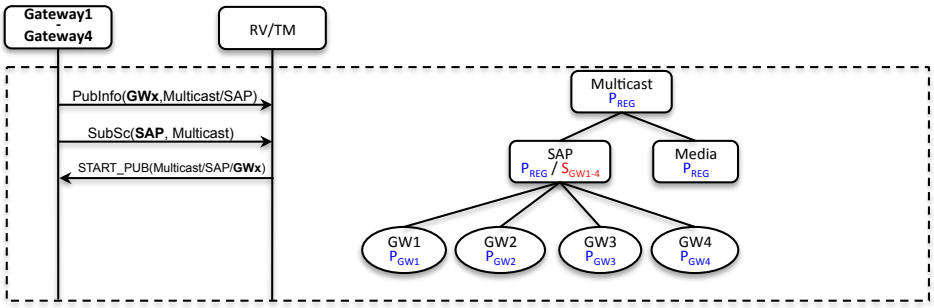


Figure 4.4: Gateway initialization for multicast support

We can see the initialization processes to support multimedia streaming over multicast in Figure 4.4 above. The gateway's first step is joining a global SAP session at 224.2.127.254. After that, the gateway publishes an information item under the ***SAP*** scope, using its own identifier as a topic, to denote that IP clients will send multicast session announcements using the SAP protocol. In addition, the gateway needs to subscribe the ***SAP*** scope in order to get information about the multimedia sessions globally. From this point, the gateway will use its information item as a topic to continuously publish SAP announcement across the pub/sub network.

In addition to the SAP announcement, our gateway joins an IGMP session at 224.0.0.22 to detect IGMP requests from the clients. It uses information in the IGMP requests to manage the group membership of the multicast session, which will be discussed later in Section 4.4.

4.3 SIP Voice Call

The users are required to make a registration to our "pubsubgateway.net" domain before initiating a SIP call. We separated the SIP voice call into two scenarios: Across the Publish/Subscribe Network and Within the Same IP Subnet. Our system is designed to handle these approaches differently and they are described in Section 4.3.2 and 4.3.3.

4.3.1 User Registration

Registration

In our example, we assume that Alice is the first SIP customer in this network. Her SIP agent registers to the SIP domain by sending a REGISTER message to the gateway's signaling port. Figure 4.5 shows Alice's registration process. Gateway1 reads the header of the REGISTER to get a username, which is Alice in this example. The gateway publishes Alice's username under its scope, ***Gateway1***. This information implies that other SIP users can contact Alice through publications and subscriptions under the ***Gateway1*** scope.

We separate SIP messages from Alice into two types: Request, and Response. The Requests are sent from Alice's UA, while the Responses are the feedback from the SIP registrar. Moreover, we include an INVITE under Alice's scope. It represents call invitation towards Alice, which the SIP registrar has to use in the call redirection. We will discuss about this INVITE in detail later in Section 4.3.2.

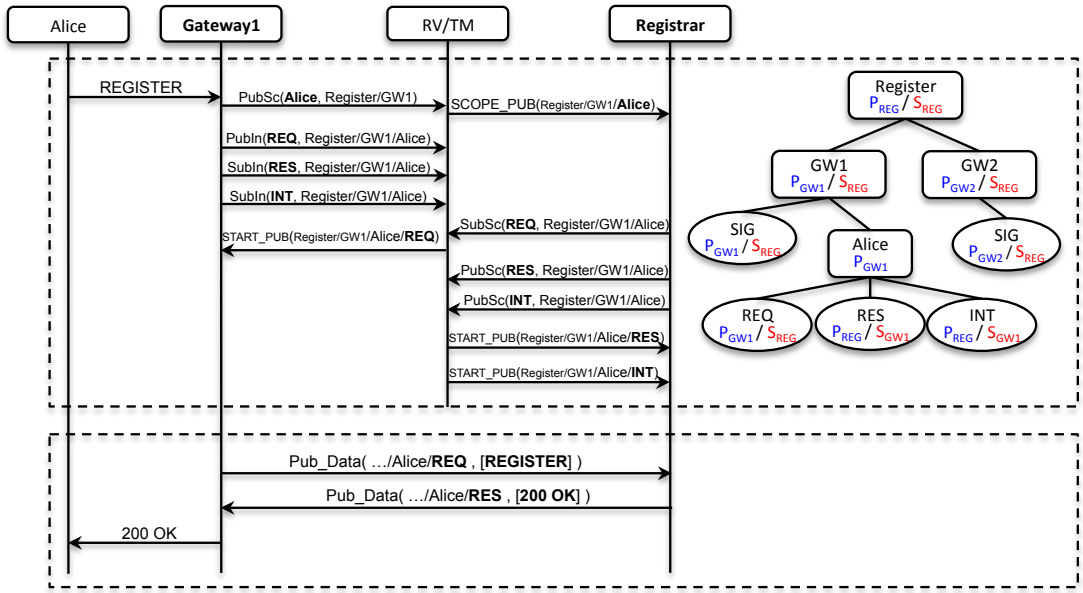


Figure 4.5: SIP user-registration via pub/sub

As we can see from Figure 4.5, Gateway1 publishes a *Request (REQ)* item to indicate that there will be SIP requests from ***Alice***. Also, Gateway1 tells the pub/sub network that it would like to get Alice's registration feedback as well as call invitations by subscribing to *Response (RES)* and *INVITE (INT)* RIDs. At the same time, our SIP registrar subscribes to the *Request* from *Alice*, and publishes *Response* and *INVITE* right after it is notified about ***Alice*** scope. The communication between Alice and the SIP registrar is established. The related information structure is presented in Figure 4.5.

Based on this information structure, the SIP registrar will get REGISTER requests and INVITE messages from all users under the ***REGISTER/<Gateway>/<Username>/****Request* topic. On the other hand, our pub/sub gateway will accept responses and SIP INVITEs through the ***REGISTER/<Gateway>/<Username>/****Response* and ***REGISTER/<Gateway>/<Username>/****INVITE* information identifiers.

The actual data exchange starts after Gateway1 receives a START_PUBLISH event for the *Request* item. This process is shown in the second part of Figure 4.5. Gateway1 publishes the REGISTER message under the *Request* RID. The SIP registrar reads the header fields of this REGISTER message to get Alice's username, signaling IP and port. Then, the registrar stores them in a location service. The format of this record is shown in Figure 4.6. After that, our SIP registrar creates a *200 OK* response to indicate that the registration is completed. It publishes this *200 OK* under Alice's *Response*, which Gateway1 later on takes charge in conveying it to Alice's UA over the IP protocol. Now, Alice is reachable via the *INVITE* item until she logs out from the system.

Username	(ip:port)	Gateway SID	CSeq

Figure 4.6: Registration table format

From the example in Figure 4.5, we have seen that the reachability is gateway specific. This comes from our assumption that the hosts in each IP network do not have any knowledge about the others, e.g., about IP addresses. Also, the SIP communication allows one user to login with several different devices, by referring to a single username. In this case, the two or more devices might have the same IP address, which cause complexity in the call redirection. Therefore, the ***Gateway*** scope allows the IP hosts in different subnet to freely use IP addresses, and helps us not only to protect the complication, describe earlier, but also to confine the traffic within only relevant area, i.e., only the network where the device is actually connected.

Deregistration

When Alice logs out, her SIP application sends a REGISTER request, but this time it comes with 0 in the Expire header. Our gateway checks the pub/sub database whether this username really exists. If it does, Gateway1 publishes this REGISTER under the *Request* topic, notifying the SIP registrar to delete the states created by Alice from the network.

The registrar unsubscribes the *Request*, and then unpublishes *Response* and *INVITE* items under the scope ***Alice***. Then, it deletes Alice's entry from its database. This triggers a STOP_PUBLISH event at Gateway1. This event signals Gateway1 to unpublish Alice's *Request*, and unsubscribe *Response* and *INVITE*. Eventually, it unpublishes the scope ***Alice***. This design choice shows that the user's states are created and deleted according to SIP signaling.

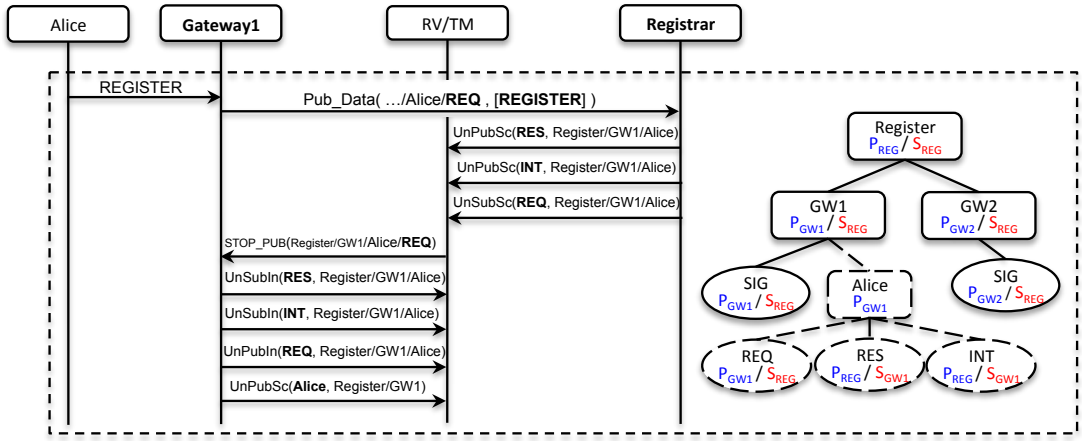


Figure 4.7: SIP user-deregistration via pub/sub

4.3.2 SIP Voice Call – Across Publish/Subscribe Network

4.3.2.1 Call Setup

Alice starts a call to Dave by sending an INVITE request to Gateway1, containing Dave's username in the *To* header. This example is presented in Figure 4.8. Gateway1's mechanisms to support this call as an originating gateway are as follows.

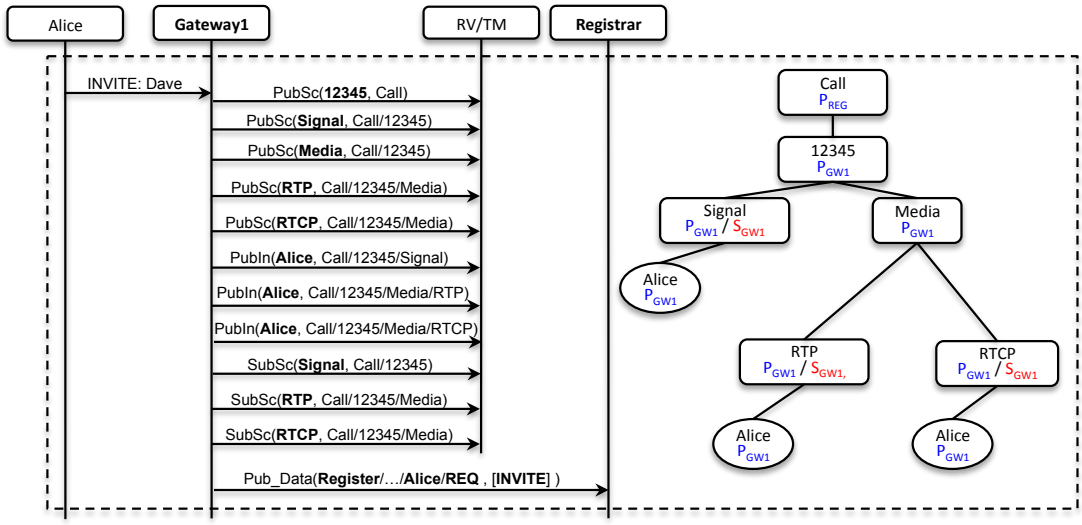


Figure 4.8: SIP call-setup at the originating gateway

1. Gateway1 inspects the INVITE packet, and publishes a call scope using Call-ID as an identifier for this specific call. The scope ID for this call is ***12345***.
2. Under the scope ***12345***, Gateway1 publishes ***Signal*** and ***Media*** SIDs to confine SIP signaling into one scope, and the media traffic into the other.
3. Gateway1 publishes scopes ***RTP*** and ***RTCP*** under the ***Media*** scope, supporting RTP and RTCP protocols from the IP network.
4. Gateway1 publishes an information item named *Alice* at the lowest level of each branch.
5. Gateway1 subscribes to scopes ***Signal***, ***RTP*** and ***RTCP***, indicating its interest to accept signaling, RTP and RTCP packets from this call.
6. Gateway1 publishes the actual INVITE request to the registrar, where the callee's username is investigated. If the callee, Dave in this example, has not registered to the domain, the SIP registrar will publish *404 Not Found* back to Alice's UA, resulting the call to be terminated.

The publications and subscriptions above generally tell us that:

- Within a SIP communication, there are signaling and media sessions.
- In the media session, RTP and RTCP protocols are utilized.
- *Alice* RIDs under ***Signal***, ***RTP*** and ***RTCP*** scopes denote that Alice is one of the call participants. Her SIP client will send and receive SIP signaling, RTP, and RTCP packets to and from the pub/sub network during the call.

If Dave is available, the information structure for SIP registration will have branches shown in Figure 4.9 as a subtree. The registrar adds Dave's IP address into the INVITE packet, and publishes this call invitation to Dave's *INVITE* RID by referring to Dave's information in the location service.

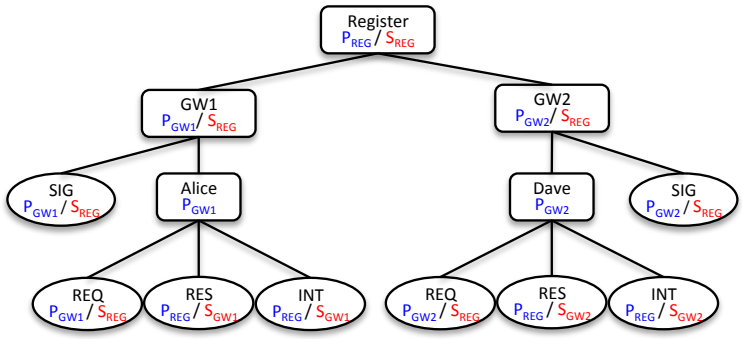


Figure 4.9: Register tree for SIP voice call

Figure 4.10 shows processes of this call setup at the destination gateway.

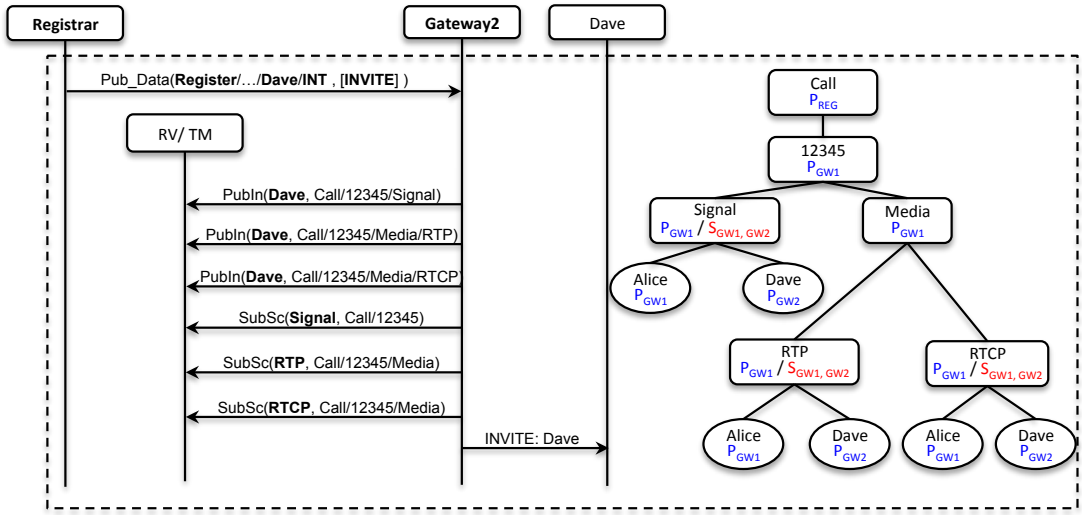


Figure 4.10: SIP call-setup at the destination gateway

1. Gateway2 knows that somebody wants to contact Dave by receiving the packet with ***Register/Gateway2/Dave/****INVITE* as a topic header.
2. Gateway2 parses Dave's IP and port for SIP signaling in the payload, and then forwards the INVITE request to that address.
3. Gateway2 reads the Call-ID from the INVITE message, and uses it as a reference to publish *Dave*'s identity under the scopes ***Signal***, ***RTP***, and ***RTCP***.
4. Gateway2 subscribes to the scopes ***Signal***, ***RTP***, and ***RTCP***, taking the role as another call participant.

Now, the information structure for the call 12345 is fully established, and ready to support interactive conversations between Alice and Dave.

4.3.2.2 Call Signaling and Media

Dave's UA sends *100 Trying* and *180 Ringing* responses to Alice, after it gets an INVITE. This is to inform Alice about the call status. Gateway2 does not have to convey these signals via the registrar anymore. It publishes all signals from Dave under ***Call/1234/Signal/****Dave* information ID, which will reach Alice through Gateway1 as Gateway1 is subscribing to the call signaling.

As soon as Dave accepts the call, the media session is established. RTP and RTCP packets from both call participants are exchanged inside the scopes ***Call/1234/Media/RTP*** and ***Call/1234/Media/RTCP***. Every time the gateway accept PUBLISH_DATA event, it forwards the actual data to the IP client according to signaling-port:***Signal***-scope, RTP-port:***RTP***-scope, and RTCP-port:***RTCP***-scope mappings, and vice versa as we can see from Figure 4.11.

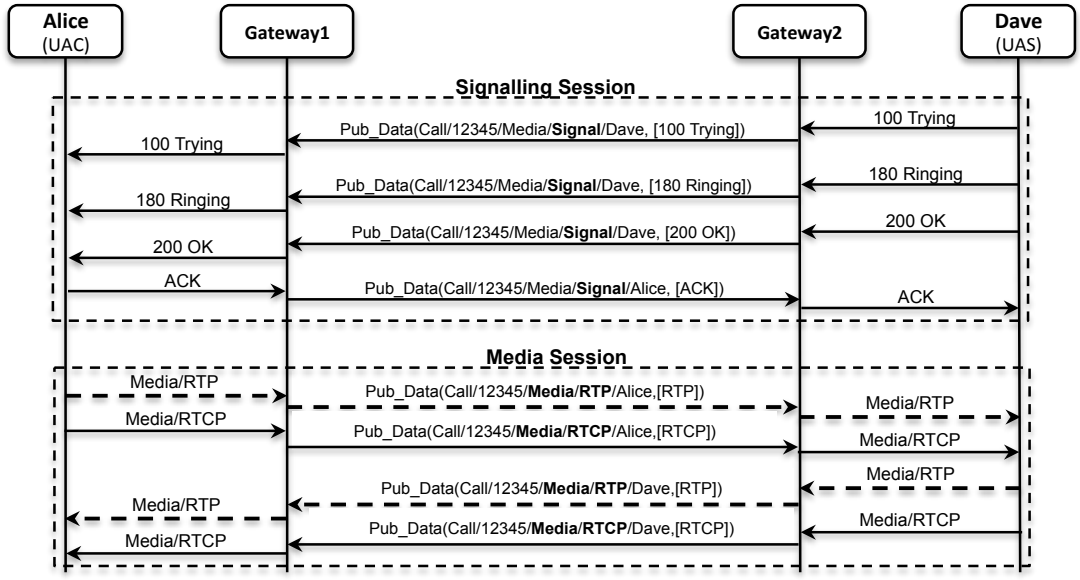


Figure 4.11: SIP signaling and media session across pub/sub network

4.3.2.3 SIP Conference

Usually, a SIP conference follows the pattern we described earlier in Section 2.4.4.5. One user agent will act as a conference-aware UA. Therefore, separate communications under two or more Call-ID scopes are established to support SIP conference. They are presented in Figure 4.12.

Alice, as an inviter of this conference, invites Dave and Carol separately. Alice arranges SIP signaling, and mixes the media before sending it to other call participants. Our gateway handles publications and subscriptions of each session in the same way described in Section 4.3.2.1-4.3.2.2.

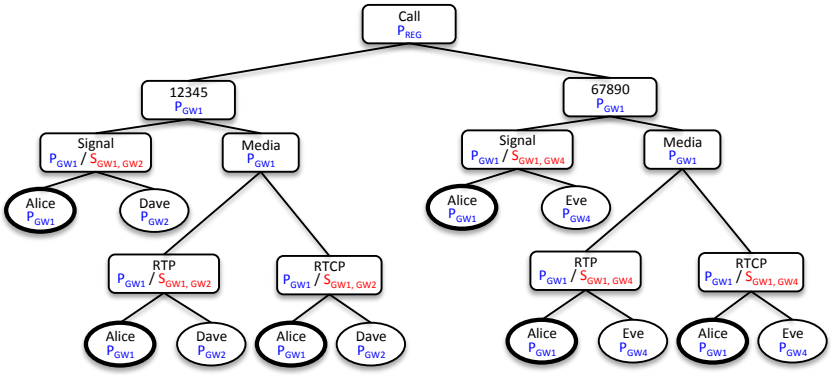


Figure 4.12: Example of an information structure for SIP conference

However, our design also supports multicast in SIP conference. Gateway1 Gateway2 and Gateway4 can use the same Call-ID as a reference. They publish one information item to indicate usernames of the call participants below ***Signal***, ***RTP*** and ***RTCP*** for outgoing packet. To receive signal and media from the call, the gateways have to subscribe these three scopes as usual. An example of the information tree to support multicast in SIP conference is demonstrated in Figure 4.13.

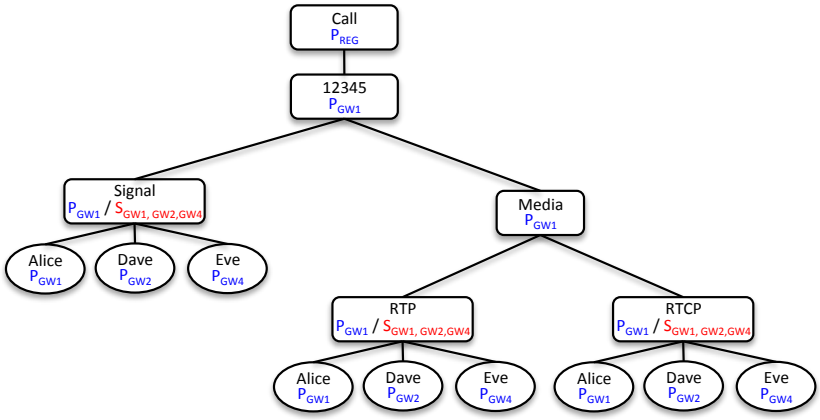


Figure 4.13: Example of an information structure for SIP conference over multicast

4.3.2.4 Call Termination

Call termination can happen either before or after the media session is established. The former case occurs when the caller sends a CANCEL request, or the callee rejects the invitation via *603 Decline* or *486 Busy Here*. If the media session has already been established, one of the SIP agents will send a BYE request to terminate the call. During the call session, pub/sub gateways inspects these signals from both IP agents, for deleting the call states accordingly.

BYE

Assuming that Alice wants to hang up an ongoing call with Dave, Gateway1 accepts a BYE from Alice, and marks in the call state that Alice wants to terminate the call named ***12345***. Gateway2 does the same when it accepts the BYE request from the pub/sub network, and it forwards this request to Dave.

At the time Dave responds with a *200 OK*, Gateway2 publishes this *200 OK* to Alice before it starts to unpublish and unsubscribe the items. In this example, Gateway2 , first of all, unpublishes *Dave* RID from ***RTP***, ***RTCP***, and ***Signal***. Then, it unsubscribes the scopes, and finally deletes call state from its local memory.

Again, the *200 OK* reaches Gateway1, which then triggers the gateway to delete all this call states from the network and its database, together with conveying the packet to Alice at the same time. Figure 4.14 summarizes gateway's action to handle the call termination in our example.

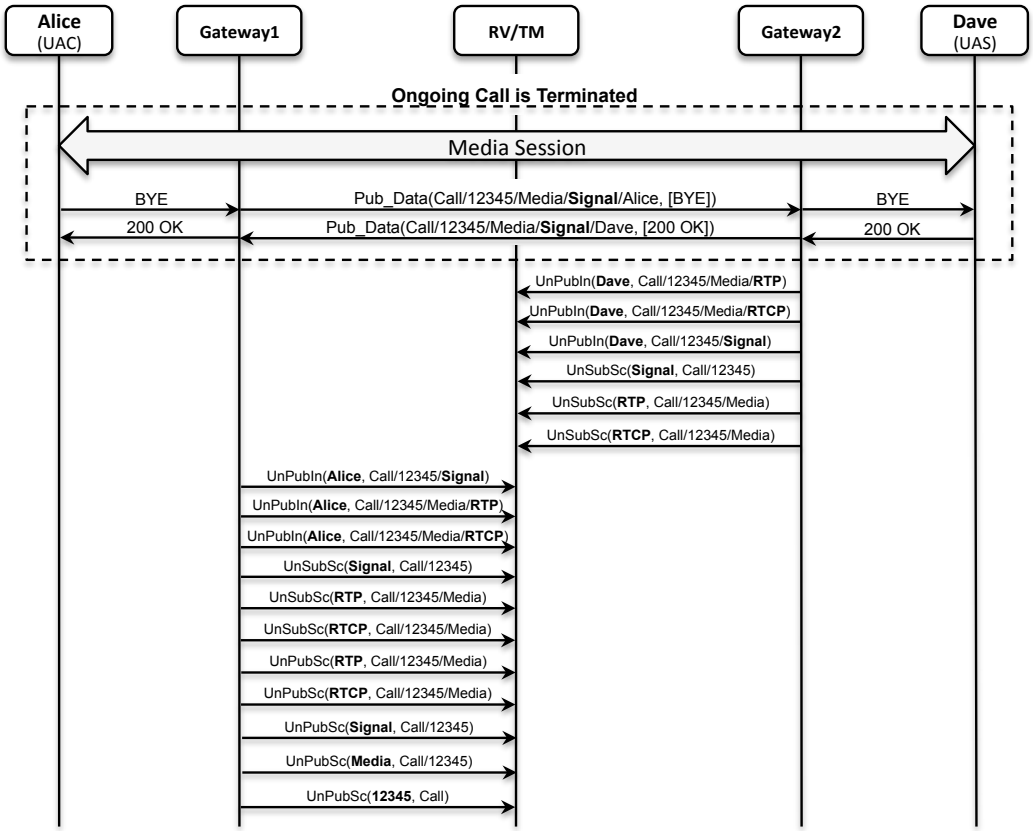


Figure 4.14: SIP call-termination over pub/sub

CANCEL, 603 Decline, and 486 Busy Here

The gateway handles CANCEL, *603 Decline*, and *486 Busy Here* in a similar way as the BYE case. Any time the gateway receives these message transfers, it marks the state of each call with that message type, and waits for an ACK from the other side. The acknowledgement is used as a confirmation to start unpublishing and unsubscribing procedures.

The procedures follow the same pattern, starting from unsubscribing and unpublishing the information items from the lowest level, and then continue to the highest scopes. Figure 4.15 demonstrates signaling flows between two gateways after Dave rejects, and Alice cancels the call respectively.

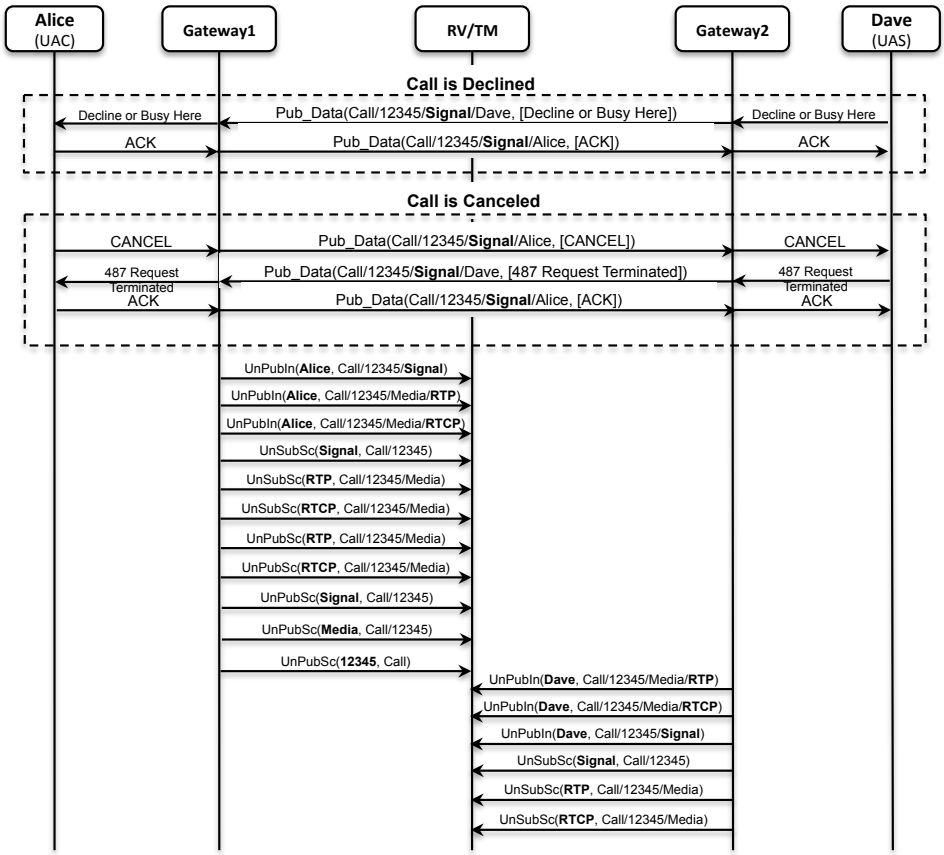


Figure 4.15: SIP call-cancelation over pub/sub

4.3.3 SIP Voice Call – Within the Same IP Subnet

For a voice call within the same IP network, our pub/sub gateway receives and publishes an INVITE message to the registrar like in the previous case. This time the gateway knows that the callee is located in the same administrative area by receiving the *302 Moved Temporarily* response from the registrar. If we consider the situation, when Carol wants to call Dave, Gateway2 will get the *302 Moved Temporarily* from the registrar. With this response, our SIP registrar attaches Dave's IP address and port in a Contact header, which helps our gateway to finally reroute the INVITE to Dave. The gateway stores the state of this Call-ID as a local call, obligating itself to use IP protocol in every packet transfer.

In fact, we firstly designed and implemented the gateway to forward the *302 Moved Temporarily* to Carol, but the SIP UA terminated the call without any reestablishment. As a result, we changed the design so that the gateway behaves like a SIP Back-to-Back user agent in this scenario. Gateway2 modifies the INVITE message so that Dave's UA communicates to RTP and RTCP ports of itself. After the call is accepted, the gateway also modifies a *200 OK* from Dave with one other pair of its own RTP and RTCP ports before delivering to Carol. This ensures that Carol's device will communicate with Gateway2 as well.

The procedure above results the media packets to pass Gateway2 during the session. The gateway inspects SIP signaling between Dave and Carol, in the same way as we have

explained in Section 4.3.2. This time, the purpose is to keep tracking the status of each call, so that the gateway can delete the local call state, without touching any pub/sub module.

4.4 Multimedia Streaming over the Multicast Technique

In addition to the SIP voice calls, our gateway supports multicast media streaming across the publish/subscribe network. This section discusses the gateway's design for this task.

4.4.1 Multicast Group Establishment

Gateway mechanisms, to support the group establishment, are depicted by Figure 4.16. We present an example, when Alice's computer acts as a media streamer, while Bob, Carol, Dave, and Eve are the members of Alice's multicast session.

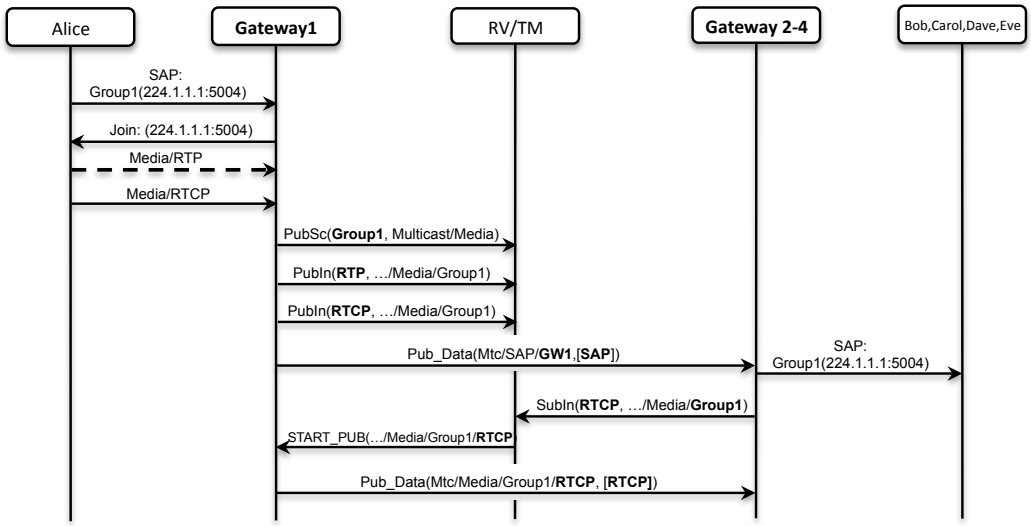


Figure 4.16: Multicast group-establishment over pub/sub

To begin with, our gateway knows when a media streaming starts from a SAP announcement, and it detects that the session is terminated by an RTCP BYE report. Therefore, the gateway is designed to join a multicast session right after it starts, in order to get RTCP packets and keep the state of each session updated according to RTCP reports.

1. Gateway1 joins multicast *Group1* after the first SAP arrives. At the same time, it creates the state of this session and publishes the actual SAP announcement by using ***Multicast***/***SAP***/*Gateway1* as an information identifier.
2. Gateway1 constructs the session's identifier by using a combination between three fields in the SDP, which are name, multicast address, and port.
3. Gateway1 publishes the constructed session identifier as a scope ID of this group under the ***Media*** scope. This SID is a representative of the multicast Group1 in the pub/sub network.
4. Gateway1 publishes two information items, which are *RTP* and *RTCP,* under the ***Group1*** scope. This indicates the support for RTP and RTCP protocols from the multimedia group.
5. Gateway2 to 4 receive the SAP packet, and create a new mapping for this session and subscribe to the *RTCP* of ***Group1***.

From this point onward, Gateway1 is a publishing gateway, which receives data from an IP network and publishes it to the pub/sub network. Also, we will call Gateway 2 to 4 as subscribing gateways, because they are responsible for subscribing information from the pub/sub network and streaming the media out to the other side via the IP protocol.

A multicast tree of this stage is presented in Figure 4.17. SAP and RTCP packets for this session are periodically multicast to other gateways until the session is terminated. At this point, only SAP announcements are forwarded to the IP subnet. The gateway drops RTCP packets, because there are no clients subscribing to this multimedia group.

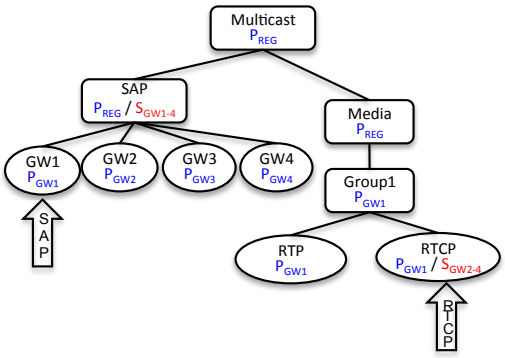


Figure 4.17: Multicast information tree

Even though this design choice creates redundant traffic load to the gateway, it gives our system more flexibility and adaptability. If the gateway does not receive any RTCP packet, it will not know when to delete multicast states from the pub/sub network and from the gateway itself, causing the states to be expanded in a long run. With this design, we can guarantee that the states will be deleted at the time the media session stops, the memory will be optimally used, and the gateway's ports are reusable.

4.4.2 Joining Multicast Group

IP hosts, such as Bob, Carol, Dave, and Eve, learn about an ongoing session from the SAP announcement. They join the session by signaling an IGMP "group join", indicating a multicast address of that session. An example in Figure 4.18 illustrates that Gateway2 subscribes ***Media/Group1/RTP*** item right after it receives an IGMP "group join" from Carol. This causes Gateway1 to start publishing the media content towards the pub/sub network.

At the same time, Gateway2 continuously accepts RTP packets of this multimedia session from the pub/sub network, and stream these media contents over the multicast address 224.1.1.1:port 5004 as stated in the SAP announcement. Gateway2 stops dropping RTCP packets and conveys them to the IP network along with the RTP packets.

The same applies to Bob and Eve at Gateway 3 and 4. As we can see from the figure, a START_PUBLISH event occurs only once, after Carol joins the session. This is because Gateway2 is the first subscriber among all pub/sub elements. For additional subscribers, the forwarding identifier will be automatically updated, so that the media RTP and RTCP are multicast to all of them.

After the media is available to each IP subnet, it is the routers' responsibility to forward the media to other additional members. Like in Dave's case, the content is available since Carol

joins the session. Our gateway only needs to add Dave in the client record without issuing any pub/sub request, while the media is forwarded to him by the IP multicast technique.

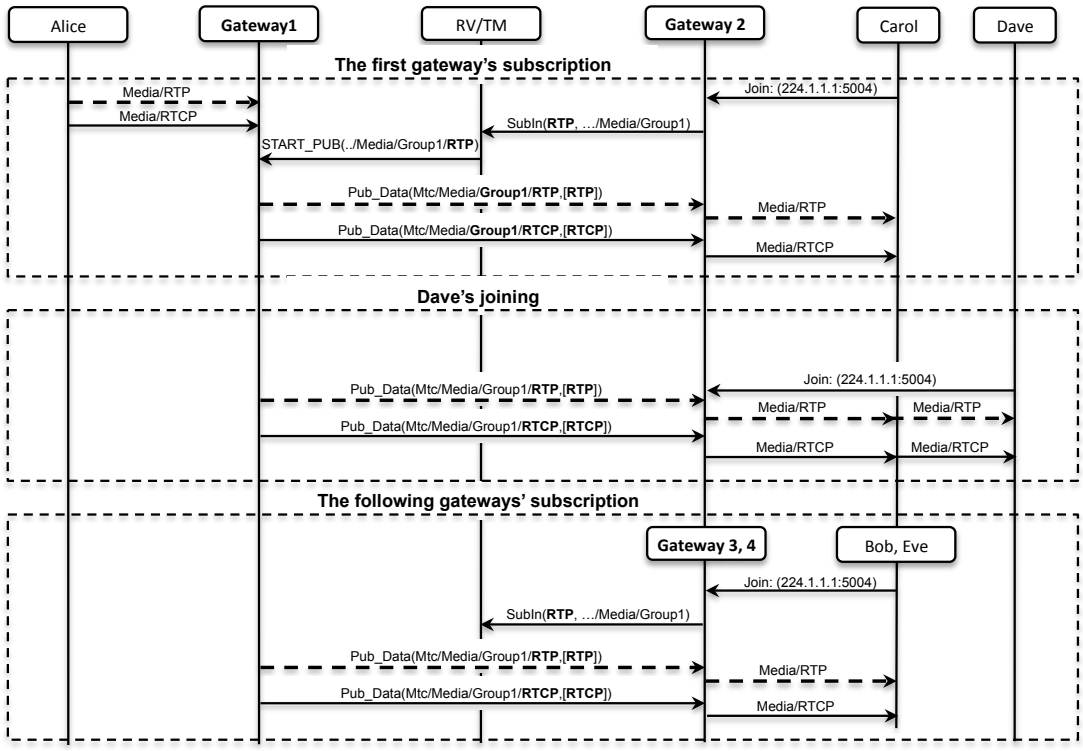


Figure 4.18: Multicast group management- Joining

4.4.3 Leaving Multicast Group

IP clients leave the multicast group by sending out an IGMP "group leave". Our gateway detects this signal and keeps the number of clients updated during an ongoing session. This information supports the gateway to unsubscribe media content from the network, when the last client leaves. From Figure 4.19, Bob and Eve are the only client. Therefore, Gateway 3 and 4 unsubscribe the ***Media/Group1/RTP*** after they receive the requests. The RTCP records still reach the gateway, but they are dropped in the same way as we have described in Section 4.1.1.

At Gateway2, IP router stops forwarding the media to Dave after he leaves the session. However, the gateway does not make any request to the pub/sub network. It just deletes Dave from the client record. Once it receives Carol's IGMP "group leave", Gateway2 unsubscribes the media *RTP* in the same way as Gateway 3 and 4. This time, a consequence is that the Rendezvous System and Topology Manager will send a STOP_PUBLISH event to Gateway1. This is because Gateway2 is the last subscriber in the pub/sub network. Finally, Gateway1 stops publishing RTP packets across the network.

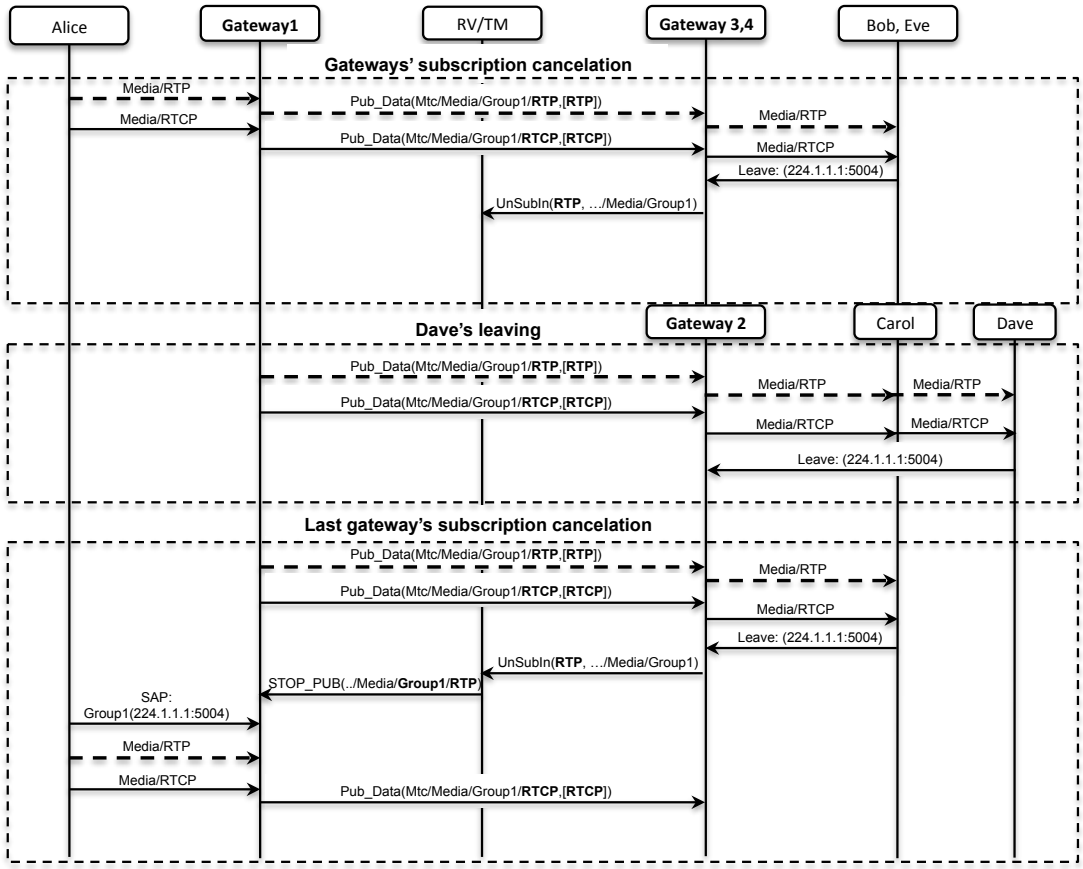


Figure 4.19: Multicast group management - Leaving

4.4.4 Multicast Group Termination

When Alice stops the media streaming, the publishing gateway, Gateway1, will receive an RTCP BYE report. It forwards this report to the network under ***Media*/*Group1*/*RTCP*** identifier, as usual. The gateway stops publishing the media and unpublishes all related items from the network. The items, that Gateway1 needs to unpublish, are *RTP* and *RTCP* information of this group and the ***Group1*** scope, as we can see from Figure 4.20.

Afterwards, subscribing gateways also receive the BYE report for terminating this group. It conveys this message to the IP network, and then unsubscribes ***Media*/*Group1*/*RTP*** and ***Media*/*Group1*/*RTCP*** in this example. Here, the state of Group1 in the pub/sub network is completely removed.

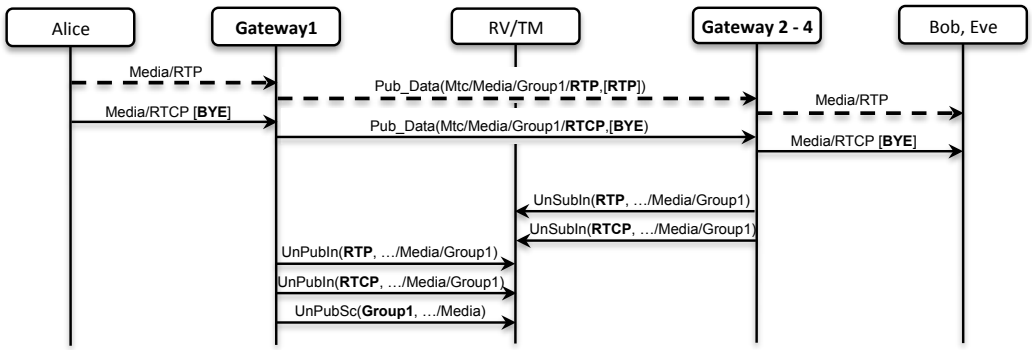


Figure 4.20: Multicast group-termination

One important detail to mention is the states in our gateways. The publishing gateway, or Gateway1 in this scenario, totally deletes the state of multicast session every time the group is terminated. This is not always the case at the subscribing gateways. The states in subscribing gateways depend on the number of IP clients. If the multicast session is terminated when there are no members left in the group, the gateway will completely delete the state.

On the other hand, if there is still some amount of active group members, our gateway keeps that state along with the client record, until the last client decides to leave. In the case that this multimedia session is re-established, our gateway will automatically subscribe to it based on the ***Group1*** SID and continue the media streaming. This design makes sure that the media will reach the same multicast address, without any additional effort from the IP clients.

4.5 Addresses and Ports Management

Apparently, the gateway has to maintain multiple connections from attaching IP clients in order to enable their communication over the pub/sub network. Therefore, we design our gateway with port management so that media from multiple sessions are independently handled.

SIP Voice Calls

SIP signaling is fixed to one port number, which is typically 5060, for every call. Two ports are assigned to each media session of the calls, one for RTP and the other one for RTCP packets. The RTP port is an even number and, the RTCP port is, as usual, its higher odd port.

Our destination gateway uses media port information in the INVITE's *m=* field, and the originating gateway uses the same *m=* information in the callee's *200 OK*. They read port numbers from the SDP, and then check whether that port is available or occupied. If it is available, our gateway will assign that RTP port along with one additional RTCP port for this session, without changing any field in the SIP PDUs. On the other hand, if the RTP port has already been occupied, our gateway chooses new port pairs, and modifies the media parameter in the SDP, before sending it to the IP client. This way, the media session will be bound to the new ports we have assigned.

Multicast Sessions

Media port management for multicast is basically the same as the one for the SIP voice call, but there is one additional parameter to consider: the IP-address of each multicast session. Anytime the subscribing gateway observes a recently established group from a SAP announcement, it compares the IP address of this multicast session to the existing ones. If this new group has the same address as an existing session, our gateway chooses a new IP address, creates the new state for this stream, and stores the recently chosen multicast address for this particular session. After that, our gateway will assign new media ports for this session, if needed. Finally, it modifies IP address and media port in *c=* and *m=* fields of the SAP announcement, to match the gateway's state every time the SAP is broadcasted.

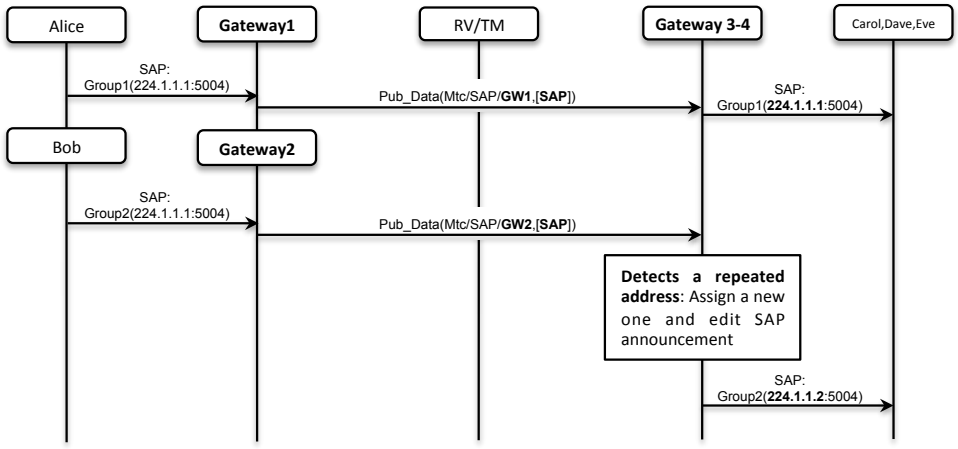


Figure 4.21: Multicast address management

As we can see from Figure 4.21, Bob streams the media to the same address as Alice at 224.1.1.1. Gateway 3 and 4 notice this anomaly by constructing the Group2 identity from SAP announcement. After that, they compare the multicast address of Group2 to the existing ones in its local memory, recognizing that the IP address is the same as the Group1. Therefore, the gateways assign the address 224.1.1.2 for Bob's session instead. Hereafter, Carol, Dave and Eve will receive Alice's and Bob's media contents by joining 224.1.1.1 and 224.1.12 without any overlapping.

4.6 Gateway Shutdown

The gateway announces that it is going to shut down by publishing a SHUTDOWN message under the ***REGISTER/<Gateway>/****SIGNAL* RID. The SIP registrar will get this signal, since it has initially subscribed to the *SIGNAL* information. Figure 4.22 illustrates an example, where Gateway4 is shutting down. After the registrar accepts the signal, it unsubscribes first the *SIGNAL* of Gateway4, and then the ***Gateway4*** scope itself. At the same time, our gateway needs to unpublish these two items. This technique enables the gateway to republish the items and repeat the system initialization process, described in Section 4.2.2, at anytime.

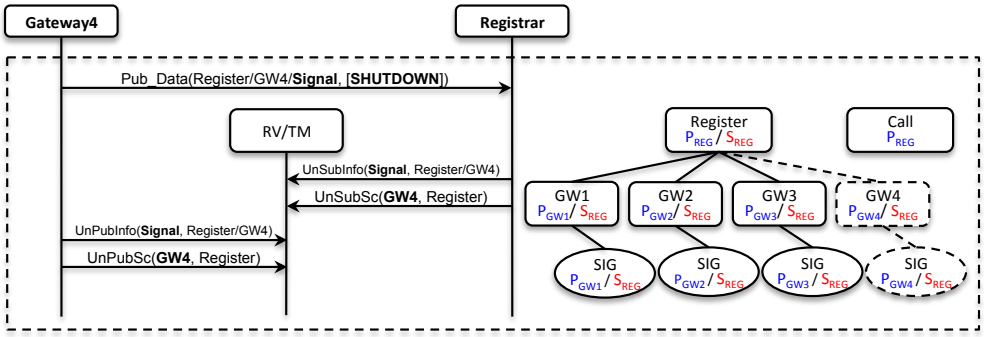


Figure 4.22: Example when Gateway4 shuts down

4.7 Summary

We presented the publish/subscribe mechanisms of gateway and SIP registrar to support SIP voice call and multimedia streaming over the multicast technique in this chapter. As the publish/subscribe system has to support the packet transfer from and to the IP clients, we designed the information structure based on the characteristic of upper-layer protocols in the current IP network, i.e., SIP, SAP, RTP and RTCP. This illustrated that each protocol contains information about the payload, and they can be implemented in the context of Information Centric Networking.

Within the signaling packets, e.g., SIP and SAP, we explicitly used unique information from these packets, as a representative of the multimedia session. This facilitates the gateway to handle each session independently by referring to the session's identity instead of acquiring them from the payload at every transfer.

In the next chapter, we discuss the system implementation in practice. We present how the components in our system-level applications are organized to fulfil the design in this chapter.

Chapter 5

Implementation

This chapter presents the prototype implementation of our publish/subscribe gateway and SIP registrar. First of all, we give an overview of the node architecture, followed by a description of the communication model of this implementation. Finally we discuss the gateway and SIP registrar implementation according to the design in Chapter 4.

5.1 Implementation Platform

Our pub/sub gateway and SIP registrar are implemented in Python (version 2.6) [49] on Linux machines. The reason why we chose Python was because it is well-known as a simple and quick prototyping tool. Moreover, it provides sufficient performance for our prototyping.

Our system is implemented on top of Blackadder API. The node architecture is illustrated in Figure 5.1. Core components of the Blackadder node consist of Click [58] elements and one additional application, taking a role as a topology manager. Our implementation uses Blackadder's API to communicate with the Click elements, signaling them to perform the tasks according to the Blackadder service model. More information about the core components can be found in [23] and [24].

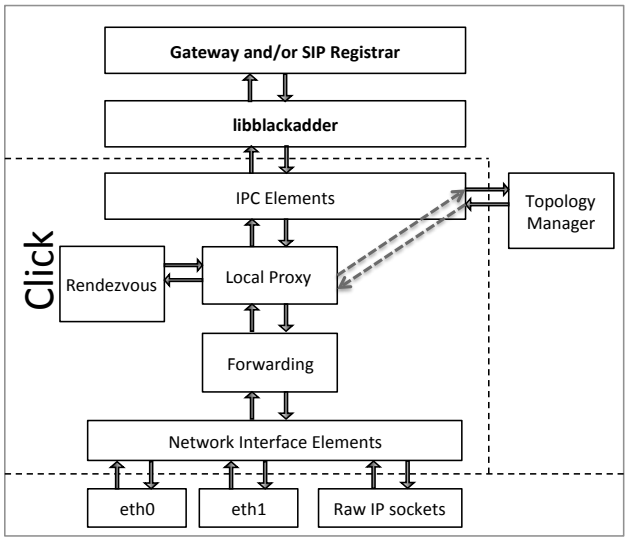


Figure 5.1: Node architecture for the system implementation

Information Identifiers

Scope and information identifiers for this prototype are fixed to 8 bytes. They are created by the SHA1 algorithm in Python's hash-library, based on the names of scopes and information items. For instance, an identifier of a REGISTER and CALL scopes, generated by this algorithm, are 0x0391D22030D15181 and 0x1B3CB484695189B5 in hexadecimal.

5.2 Communication Model

Figure 5.2 depicts the communication model of our system. All nodes are physically connected by Ethernet links. Shaded areas in the figure represent pub/sub components. The uppermost components are our applications, and other lower white layers are IP components. As we can see, the publish/subscribe gateway communicates with IP components through the socket API, while it contacts SIP registrar and other pub/sub elements through the Blackadder API. As a result, the interface that connects to IP network needs to be configured with an IP address.

In legacy networks, we utilized the UDP protocol over IP sockets to send and receive SIP [29], SAP [37], RTP and RTCP [19] datagrams. Besides, the gateway handles IGMP records [5] through raw sockets, since the IGMP is a Network Layer protocol. The transport channel for SIP signaling is port 5060. In addition to this, our gateway acquires SAP announcement and IGMP membership report by joining multicast sessions at 224.2.127.254 port 9875 and 224.0.0.22 respectively. Apart from that, we have implemented the gateway's mechanism to assign private ports for RTP and RTCP flows when needed.

The communication strategy in the pub/sub network is DOMAIN_LOCAL. We chose this strategy because it enables the communication between every pub/sub node in the network. A publisher publishes data using a *publish_data* function. At the subscriber's end, the data arrives with a PUBLISH_DATA event. Together with the actual data, our system knows the type of data by examining the event's topic identifier.

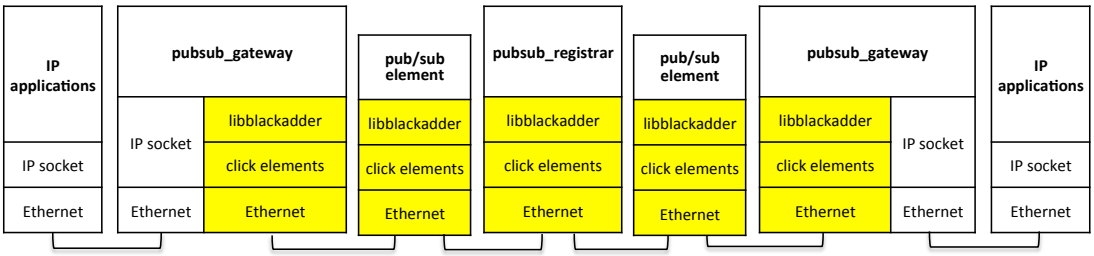


Figure 5.2: Communication model of the prototype implementation

5.3 Implementation Architecture

In this subsection, we present the architecture of our prototype. Publish/subscribe gateway and SIP registrar were implemented as individual applications. Furthermore, inside the gateway's architecture, there are two independent modules, called SIP and Multicast.

5.3.1 Publish/Subscribe Gateway Implementation

First of all, our gateway needs to be pre-configured, in order to know which interface is connected to the IP network. The gateway has to retrieve an IP address of this interface, and use it in a DNS response. The gateway's operation for DNS support will be explained later

in the next subsection. The IP interface is given in a command-line option, such as “eth0” if Ethernet 0 connects to an IP subnet.

The SIP and Multicast modules are multi-threaded. Even though the gateway is divided into two separate parts, they share one common component named ***pubsub_handler***. It is an interface between the gateway application and the Blackadder library, enabling communication between our gateway(s) and registrar applications. Other than executing pub/sub commands, this component lists all publications and subscriptions into a hash table. The purpose of this is to use the records as a reference to unpublish and unsubscribe information afterwards. An example of the records’ use case is when Alice logs out off the domain. Gateway1 unsubscribes, and then unpublishes all her information according to the ***Alice*** scope in the record. This procedure corresponds to Figure 4.7. After that, the *pubsub_handler* deletes Alice’s entries from the hash table, ensuring that there is no state left in neither the network nor the gateway’s local memory.

One other responsibility of the *pubsub_handler* is to acquire pub/sub events, specified in Section 2.1.1.1. If the event is either START_PUBLISH or STOP_PUBLISH, this *pubsub_handler* will send an order to a responsible module to start or stop the publication. The actual data comes with the PUBLISH_DATA event. Our *pubsub_handler* reads the event topic and relays it to a responsible module, where the data will be processed later on.

5.3.1.1 SIP Module

The SIP module is divided into seven components as shown in Figure 5.3.

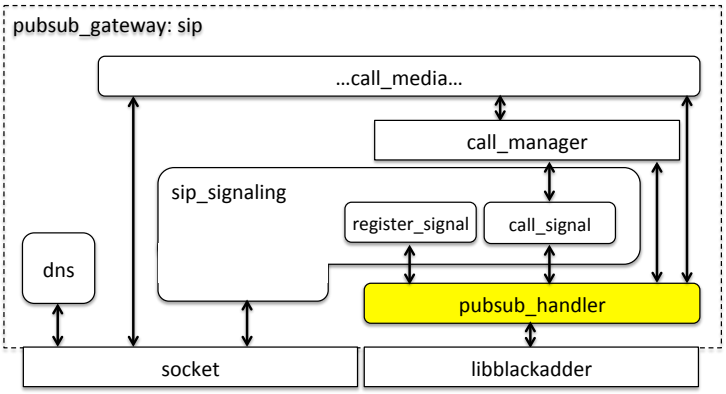


Figure 5.3: Implementation architecture for pub/sub gateway- SIP Module

DNS Thread

DNS is an IP thread, which receives DNS queries from the IP hosts. If a SIP agent sends the query for the “pubsubgateway.net” domain, this module responds by attaching the address of the IP interface in the DNS’s ANSWER field. After this, the hosts will use this encapsulated IP address to contact our pub/sub domain.

Sip_signaling Thread

Sip_signaling is another thread that continuously listens to SIP messages, from both IP and pub/sub networks. We distribute the workload of this thread by classifying the message exchanges into two categories: SIP Registration and SIP Call. The thread differentiates them from the Method in a SIP Request-Line, or in a response’s CSeq. It dispatches the request with REGISTER method to a *register_signal* thread, and others to a *call_signal* thread for

further processing. Additionally, the *sip_signaling* has to forward SIP messages from the *register_signal* and *call_signal* threads to the host with a socket API.

Register_signal Thread

The *register_signal* thread takes care of user registration according to the design in Section 4.3.1. It collaborates with the pub/sub registrar, and performs the tasks according to the requests and responses. Any time the thread receives a REGISTER message from the new user, it creates a pub/sub identifier from the username, publishes it as a scope, and exchanges requests and responses under *Request* and *Response* IDs. Then, the thread publishes the actual message under the ***Register*/*<Gateway>*/*<Username>*/*Request*** topic as shown in Figure 4.5. At this stage, the thread temporarily stores a mapping between the pub/sub identifier and user's IP address. It waits for the registrar's response and uses the address to route this response back to the user's device. This registration entry will be removed after the *200 OK* for each respective request is routed back.

This *register_signal* thread can also detect a deregistration request. This time, it directly publishes the request to the network, and again waits for a STOP_PUBLISH event of the user's *Request* item. This event notifies the thread that the registrar has already removed the user's information. It allows the *register_signal* thread to unsubscribe *Response* and *INVITE*, as well as unpublishes ***Alice***'s *Request* and scope as shown in Figure 4.7.

Call_signal Thread and Call_manager Component

The *call_signal* thread and *call_manager* component handles call setup, cancellation and termination both locally and across the publish/subscribe network. The *call_manager* is an interface between the *call_signal* and *call_media* threads. It stores active call states in the database, using the formats shown in Figure 5.4 and Figure 5.5. We utilize Python dictionaries to store the call states in this case. It allows us to use a pub/sub *Call-ID* as a key reference to access other data in the hash, and refers to the ***Call-ID*** scope in the pub/sub network at the same time.

Call-ID	Client's username	Client's signaling address	Client's RTP address	Client's RTCP address	Gateway's RTP port	Gateway's RTCP port	Media thread

Figure 5.4: SIP global call-table format

Call-ID	Caller signaling address	Callee signaling address	Caller RTP address	Callee RTP address	Caller RTCP address	Callee RTCP address	Gateway's RTP ports	Gateway's RTCP ports	Media thread

Figure 5.5: SIP local call-table format

The local table contains information for both caller and callee, because the gateway has to communicate with two IP agents at the same time. On the other hand, our gateway communicates with one client in the global situation. This makes it necessary to store the information from an adjacent client only. The client's signaling, RTP, and RTCP addresses are <IP:port> tuples. The signaling address is a return value from the socket API, whereas the client's RTP and RTCP ports are retrieved from an *m=* field in the SIP's SDP. An originating gateway, e.g., Gateway1 in Alice-to-Dave example, learns Alice's information from an INVITE signal. Besides, a destination gateway, which is Gateway2, gets Dave's RTP port from the *200 OK*.

The *call_manager* assigns two additional ports for the gateway's *call_media* thread. These ports are specifically bound to the call's media session. The thread uses them to send and receive voice packets to and from the clients above. Finally, the *call_manager* receives an order from the *call_signal* thread to create a new call entry, new media thread, to start, stop and delete the entry.

The *call_signal* thread executes the orders according to the actual SIP messages. It signals the *call_manager* to create the new call entry, when it recognises an INVITE message, as well as publishing and subscribing call-related information, as shown Figure 4.8 and 4.10. Additionally, the *call_signal* thread sends the *call_manager* the orders to create the *call_media* thread when it recognises a *200 OK* of the respective invitation, to start/stop the media thread when it receives the START_PUBLISH/STOP_PUBLISH events from the *pubsub_handler*, and finally to delete the entry when it detects a BYE, CANCEL, *603 Decline* or *486 Busy Here*.

Call_media Thread

The number of *call_media* threads depends on the amount of active sessions. There are two types of the threads in our implementation: Local and Global media threads. Both of them are responsible for the call's media session. The local thread uses RTP and RTCP protocols to communicate with both call participants as illustrated in Figure 5.6. It receives voice packets from Carol through one RTP and RTCP ports of the gateway, and then forwards them to Dave through the other port pair, and vice versa.

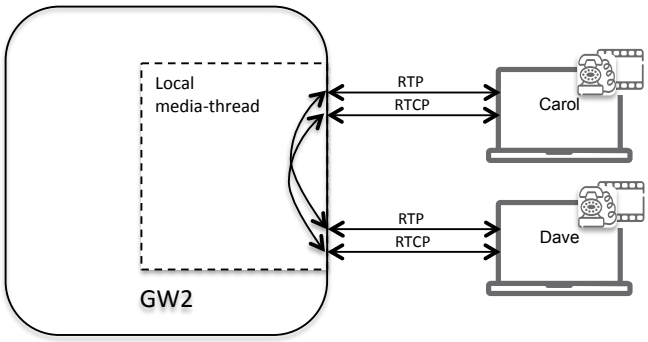


Figure 5.6: Local media-thread implementation

In the global case, two threads have to work synchronously to enable bi-directional conversation across the network. These threads subscribe to the same ***RTP*** and ***RTCP*** scopes of the call, presented in Figure 4.10, right after the *call_manager* creates them. They receive the voice content through the socket API, and publish the payload under the media scopes. Conversely, the threads forward media packets to the IP client, anytime they accept the publication from the pub/sub network.

In Figure 5.7, we can see that the media-thread at Gateway1 receives RTP and RTCP packets from Alice. It publishes the packets under ***Call/12345/Media/RTP***/Alice and ***Call/12345/Media/RTCP***/Alice separately. The publications arrive at Gateway2 since the thread has subscribed to the media scopes. Gateway2, later on, forwards this data to Dave using an IP protocol. Similarly, the thread at Gateway2 receives Dave's packets through the socket API, but this time it publishes data under ***Call/12345/Media/RTP***/Dave and ***Call/12345/Media/RTCP***/Dave topics. Finally, Gateway1 accepts the publication and forwards it to Alice.

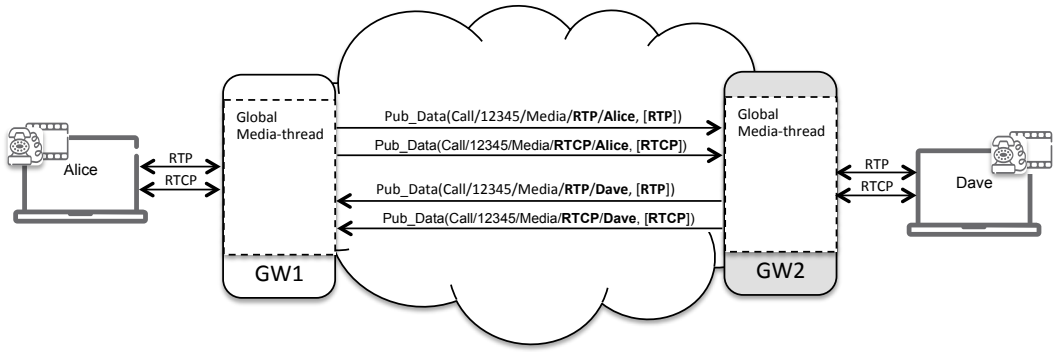


Figure 5.7: Global media-thread implementation

5.3.1.2 Multicast Module

Next we take a look at the multicast module. ***Pubsub_handler*** in the SIP registrar works in exactly the same way as the gateway's, except that it has to take into account the SCOPE_PUBLISHED and SCOPE_UNPUBLISHED events, as well. Apart from the *pubsub_handler*, this module has three different threads, and one submodule for group management called a ***group_manager***. The responsibility of the *group_manager* is similar to the *call_manager* in Section 5.3.1.1. It accepts commands from *sap*, *media_streamer*, and *igmp* threads to create or delete multicast state, and to start or stop the *media_streamer* respectively. The architecture of this module is presented in Figure 5.8.

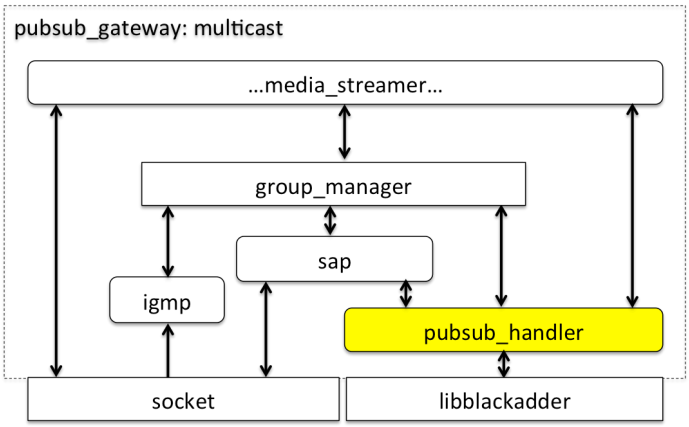


Figure 5.8: Implementation architecture for pub/sub gateway- Multicast Module

Sap Thread

The *sap* thread starts to work by joining a SAPv1 session at 224.2.127.254 port 9875. If there is an announcement for a new group, the thread will signal the *group_manager* to create a mapping, following the format in Figure 5.9. Group-ID is a pub/sub unique identifier for each multicast session. When the *sap* thread receives a SAP announcement from the IP socket, it publishes *RTP* and *RTCP* items of a new multicast session in Figure 4.17 by using this Group-ID as a scope. Then, it publishes an actual SAP announcement under the ***Multicast/SAP/***<*Gateway*> topic. The Gateway's status is set to PUBLISHING mode.

Group-ID	Multicast (address:port)	Multicast name	Gateway's (RTP, RTCP) ports	Media streamer thread	Gateway's status	Client list

Figure 5.9: Multicast group-table format

Multicast <address:port> and name are the fields, telling the actual information about the session. The clients indicate this <address:port> to join or leave the group. Next, the *group_manager* assigns two available ports at the gateway to send and receive the media stream. They are stored in the gateway's RTP and RTCP ports entry.

From a receiving gateway's point of view, the *sap* thread accepts an announcement from the *pubsub_handler*. It sends the *group_manager* an order to create the state and a new *media_streamer* thread with the format in Figure 5.9 as well. But this time, the Gateway's status is set to SUBSCRIBING mode.

Igmp Thread

The Client list in Figure 5.9 stores IP addresses of the groups' participants. The *igmp* thread manages this list by listening to an IGMP report. It signals the *group_manager* to add an IP address of the client when the client sends an IGMP "group join" message. If the *igmp* thread detects a "group leave" report, it will send the *group_manager*, a command to delete that IP address from the list. The *group_manager* checks the amount of participants every time it gets an order to update the client list. This is for scheduling the *media_streamer* to start working when the number of client goes from 0 to 1, and to stop after the number reaches 0.

Media_streamer Thread

The hash table in Figure 5.9 points to a responsible *media_streamer* thread, which can operate in either PUBLISHING or SUBSCRIBING mode from the pub/sub network point of view. In the PUBLISHING mode, *media_streamer* receives the media via an IP protocol and then publishes it using the publish/subscribe paradigm. On the contrary, the thread in SUBSCRIBING mode behaves like a media streamer in the IP subnet. It receives the media from the pub/sub side and streams it over an IP multicast on the other side.

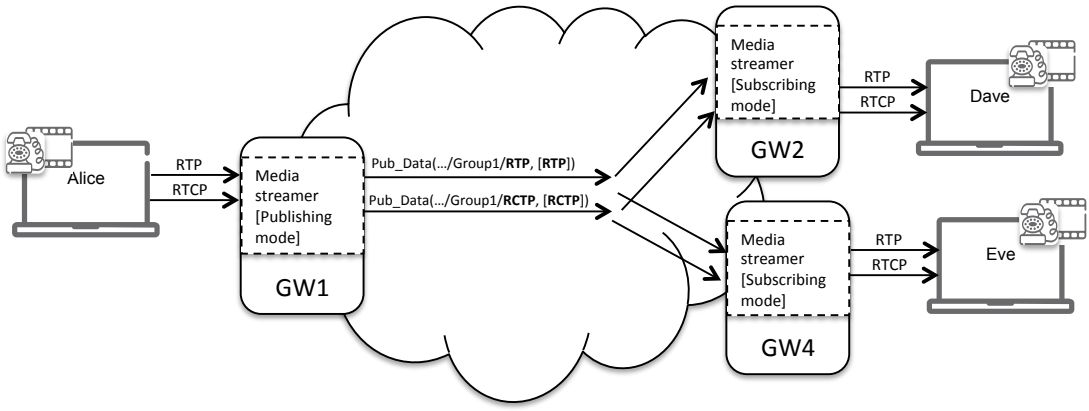


Figure 5.10: Media streamer-thread implementation

Figure 5.10 depicts the cooperation between the *media_streamer* threads to support multimedia streaming. The threads in PUBLISHING and SUBSCIBING modes work synchronously by pointing to the same pub/sub ***Group-ID***. The threads start to work right

after the state is created. This can be referred to the design in Section 4.4.1. The thread in PUBLISHING mode periodically publishes RTCP reports to the network from the beginning, but it will not publish the media content until it gets the START_PUBLISH event for the *RTP* topic. Moreover, the thread will stop the RTP session if it accepts a STOP_PUBLISH signal.

The *media_streamer* thread in SUBSCRIBING mode works reversely. It receives RTP publications from the PUBLISHING mode thread, and then sends the packet out to an IP network using the multicast address. The thread knows this address from the mapping shown in Figure 5.9.

Ultimately, if the *media_streamer* recognizes an RTCP BYE report, the thread will immediately stop and notify the *group_manager* to make a decision whether to keep or delete the state, in accordance with the design in Section 4.4.4.

5.3.2 Publish/Subscribe based SIP Registrar Implementation

The SIP registrar is a pure pub/sub component. Its architecture is shown in Figure 5.11. The registrar communicates with our gateway through the ***pubsub_handler***. Incoming publications are classified into two categories: User Registration and Call Redirection. The *pubsub_handler* forwards the user registration messages to a ***register*** thread. If the message is a registration request, the *register* thread will stores the user's information according to Figure 4.6 in a ***location service***, and then publishes a *200 OK* back under the user's *Response*. On the contrary, when the thread accepts a deregistration request, it deletes that particular user's entry, as well as unpublishes and unsubscribes the pub/sub states.

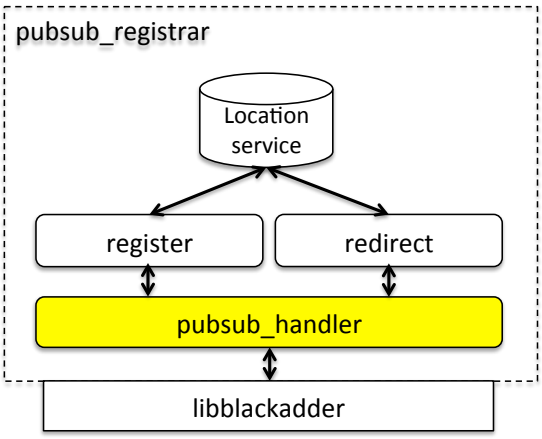


Figure 5.11: Implementation architecture for pub/sub based SIP registrar

The location service is also accessible by a ***redirect*** thread. This thread is responsible for redirecting a call invitation to the destination gateway. As soon as it gets an INVITE, the thread checks the call recipient's availability and other related information from the location service. If the recipient's information is not in the database, this thread will publish *404 Not Found* back to the caller's response scope. Otherwise, it adds the callee's <IP:port> to the INVITE message, and finally publishes this INVITE to the destination gateway.

One other possible response from the *redirect* thread is a *302 Moved Temporarily*. The thread publishes this response, when it finds out that the Gateway SIDs are the same for both call originator and receiver. This notifies the gateway to handle the call locally, as mentioned in Section 4.3.3.

5.4 Summary

The gateway implementation composes of two separate modules: SIP and Multicast. The SIP module together with the SIP registrar implementation is used for supporting the voice communication. The SIP registrar implementation meets the basic requirements stated in the RFC 3261. Within the scope of SIP voice call, we assigned one thread to handle the user registration and the other one for the call signaling, in both the gateway and SIP registrar. This is because these two processes work independently.

The SIP call can be separated further into two types, which are the local call and the call across the pub/sub network. Even though the content of call signaling and media are the same, our gateway utilizes different mechanisms to forward the packet. The threads use only socket API if the call happens within the same IP subnet, while using both socket and Blackadder API for the call across the pub/sub network.

We implement the Multicast module for our pub/sub gateway to support globally announced multicast session. It conveys SAP announcement as well as the media content to and from the pub/sub network. We, again, separated the direction of the media content into two types. The first one is to receive media from the IP socket, and publish it to the network. The other one is to acquire the PUBLISH_DATA event and stream it through the IP socket when necessary.

In Chapter 6, we demonstrate the performance of our system implementation, starting from the architectural setup of the prototyping network. Then, we describe testing and measurement procedures followed by the result and analysis.

Chapter 6

Evaluation and Discussion

In this chapter, we first describe the testing environment, used for experimenting the implementation of both pub/sub gateway and SIP registrar. The evaluation is separated into three major approaches: functional testing, traffic-based analysis, and time-based analysis. The functional testing presents how well the implementation fulfils the requirements stated in Chapter 3. In the traffic-based analysis, we observed real-time multimedia communications using publish/subscribe paradigm in multiple stages. Then, we made a comparison of the call setup duration over pub/sub and the IP network in the time-based analysis. Section 6.2 explains how the testing and measurements of these three approaches were performed. Finally, all of the results are presented and analysed in Section 6.3.

6.1 Testing Environment

The purpose of creating a testing environment was to simulate the migrating phase, where the publish/subscribe network is used as the Internet, and the IP legacy is still used in the client's network, e.g., at home or in the office. Our gateways were installed at the edge of the pub/sub network, connecting the IP and publish/subscribe networks together. This enabled IP-based clients to communicate with each other through the pub/sub. Secondly, the testbed was used for examining the performance of our system implementation in different circumstances and levels of complexity.

The testbed consisted of 41 virtual machines as presented in Figure 6.1. Within these virtual computers, 14 of them were representing the publish/subscribe network, and the other 27 machines were used as the IP hosts. The operating system of the virtual machines was Linux version 10.04. The kernel version for all pub/sub elements was 2.6.32.

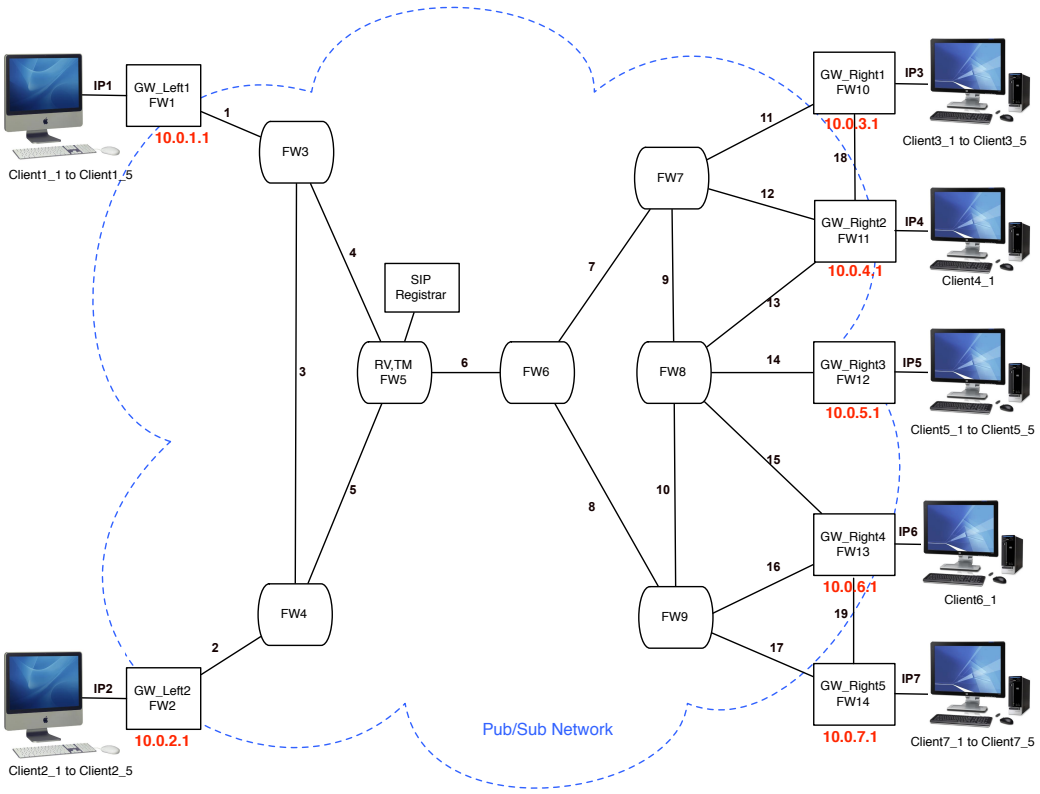


Figure 6.1: Network topology of the testing environment

The publish/subscribe operations were performed by the PURSUIT's Blackadder prototype version 0.2.1, running on the user level. The Blackadder prototype took care of all basic signaling within this pub/sub network. Each of the nodes inside our pub/sub network was called a "Forwarder". Forwarder5 was chosen to be both Rendezvous node and the Topology Manager of this prototype. It handled data publication, subscription and manipulated the paths between the hosts. SIP registrar was also running on top of the Forwarder5, and Forwarder 1, 2, 10, 11, 12, 13, 14 were chosen to be the gateways. IP addresses were assigned to all of the gateways in our testbed, and they are stated under the gateways in the Figure 6.1.

There were 19 pub/sub and 7 IP links in our prototyping network. All of the links were numbered as shown in the figure. These numbers will be used later in the subsections for more visible explanation.

At the client sides, all virtual machines had a SIP application called "Blink" [7], and VLC media player version 2.0.1 [60] for the multimedia streaming test. These applications were downloaded from [4] and [59]. Regarding the IP network area, there were five clients connecting to gateways Left1, Left2, Right1, Right3 and Right5, while the other two were connecting to gateways Right2 and Right4 respectively. All of the clients were used as call initiators, call receivers, multicast streamers, and the members of multicast sessions.

Generally, client registration had to be preformed at the beginning of the SIP-based test. We observed how the registration process was accomplished. Finally, we generated calls between each of the user agents. In the multimedia-streaming test, the multicast session was

generated from one of the hosts, acting like a media server. The information related to the particular multicast session was announced by SAP protocol. After that, we examined how the session was joined and left by the clients.

The number of SIP and multicast sessions could be one or multiple at a time, depending on the goal of each test. In all cases, we used one evaluation program, written in Python, to monitor the traffic at every link. This script differentiates IP and Blackadder traffic from each other, as well as detecting different protocols, types of signal, traffic originators and destinations. This information were structured and printed into a text file, following the format in Table 6.1. More specific testing procedures and measurements are explained in the following sections.

Table 6.1: Output format from the evaluation program

Time	IP or Pub/Sub	From Mac Address	To Mac Address	Source IP	Destination IP	Packet Length	Packet Overhead	Pub/Sub Event Type	RID

6.2 Evaluation Procedure

This section explains how the experiment was performed in each category. For the functional testing, all of the IP hosts took part in the experiment, whereas Gateway Left1, Left2, Right1, and Right2 were used in the traffic and call setup measurements.

6.2.1 Functional Testing

The functional testing was carried out to observe the behaviour of our gateway and SIP registrar implementation. We assumed that the SIP registrar was always started before the gateways. Following this, we examined how efficient the implementation could handle different network situations, and how it was aligned with the requirements of this thesis.

The very first state to discover was the system initialization, i.e., when the gateway started to operate. Starting from this phase, we analysed how the gateway was functioned, and did the experiments, described in the following subsections. After each aspect of the experiment was accomplished, we gracefully shut down the gateway and reinitialized it. This was to ensure that the gateway could come back into operation at any time.

Regarding the result of this functional test, there were common aspects that we observed, examined and verified in each step. From the pub/sub network point of view, we examined how the pub/sub gateway manipulated the information structure by seeing the output from the rendezvous system. This included data publications and subscriptions from all relevant participants. After the publication and subscription was completed, we concentrated on the data transfer, seeing that SIP, SAP, RTP and RTCP packets had been published under the corresponding scopes and items as designed.

At the IP client, we analysed the message exchange between our gateway and the host. We detected SIP sequences, looked at SIP and SAP headers, together with its SDP. The reaction from SIP application and the VLC was another alternative to confirm the correctness of our design. We verified that the applications reacted as we had expected. For instance, the SIP application should terminate the call and show the reason accordingly. In addition, we repeatedly observed the state of each session inside the gateway. This was for evaluating how efficiently things were managed in our design.

6.2.1.1 SIP Registration

SIP Registration is an initial operation, performed by all clients, before the calls were possible. We tested it by making a SIP registration from all of the hosts in the network. Then, we observed the registration entries in the SIP registrar, at each time the REGISTER message arrived. We also examined the users deregistration process, when we closed the SIP application, or logged out from our SIP domain, in order to see the correctness of the registration feature in the SIP registrar implementation. It should parse, store, and delete the right piece of data as needed.

6.2.1.2 SIP Voice Call

We started this test by making a call between two clients behind different gateways, followed by a call between the ones within the same IP subnet. At this stage, we observed the gateway's mechanism to support the call and verified whether the mechanism was precise enough. Once the SIP call was initiated, the experiment was performed to simulate multiple reactions towards the call. These are when the callee accepts, or declines the call. In addition, we simulated the possibility for the call to be cancelled before any decision was made by the callee. If the call was accepted, we tried to hold the session, resume it, initiate a SIP conference, and eventually terminated it from both sides of the connection.

6.2.1.3 Multicast Media Streaming Over Publish/Subscribe Network

The experiment began when Client1_1 started to stream a sample video to the multicast address, for example 224.1.1.1, using the RTP protocol. SAP message was announced to the network. After that, all clients available in the testbed joined the multimedia session. Here, we captured the SAP announcement at every host with Wireshark [42], saw the property of an SDP inside the packet, and considered the completeness of the video.

After all clients had been participating in the session for some period of time, they started to leave. One client was scheduled to leave the multimedia session at a time until there were no clients left in the session. At this stage, we focused on the gateway's capability to unsubscribe and unpublished information in accordance with the design in Chapter 4.

6.2.1.4 Reachability Testing

We tested that the SIP session can be created from and to any node in the network by performing the test in Section 6.2.1.2. We initiated the calls from every IP client towards the hosts behind every of the gateways, including the ones in the same subnet. We verified the result of each experiment by checking whether the call could reach all destinations as expected.

The test for multimedia session was done in a similar way. We created the multimedia sessions from every gateway, and joined them from the hosts in all other IP networks. We determined that the video could reach the hosts as planned.

6.2.1.5 Support for Concurrency Testing

The scalability support was tested by generating multiple SIP calls and multimedia sessions at the same time. The calls were made among all the gateways, and the videos were streamed from multiple hosts. In addition to this, calls and multimedia were randomly initiated during some period of time, meaning that the sessions were generated and terminated at an irregular time frame.

6.2.1.6 Adaptability Testing

In this phase, we intentionally configured the SIP destinations behind the gateways, other than Gateway_Left1, to use the same RTP and RTCP ports. Then the call was made from the hosts behind Gateway_Left1 to all of the others.

The multicast streaming worked differently in this task. We tested the adaptability of our design by sequentially streaming the video from the hosts behind all gateways via the same multicast IP and port pair. First, we captured the SAP message and checked the SDP in its payload. After that, we joined all multicast sessions available and verified the accuracy of the implementation.

6.2.2 Traffic Measurement

This part of the measurement was to study amount of traffic inside the pub/sub network, at different stages of the communication. We separated the traffic measurement into two sessions. In each session, the traffic was classified as shown in Table 6.2.

Table 6.2: Phases of the traffic measurement

SIP Session	
User Registration	- SIP user starts the application and activate a SIP account
Call Signaling	- SIP user initiates and terminates the call
Multimedia	- The multimedia session is established, and the conversation is taking place
Multicast Streaming Session	
SAP Announcement	- Server starts the media streaming, without any subscription from a client. Multicast session is announced through SAP only.
Session Joining	- Multicast session is joined by the first client, across the pub/sub network. - Multicast session is joined by several clients behind the same gateway - Multicast session is joined by several clients behind different gateways
Session Leaving	- Client(s) starts to leave the multicast group - After the last client leave the multicast session

The experiment was carried out in an order presented in Table 6.2. The traffic was captured at every pub/sub node using the *ifstat* [21] command. The *ifstat* captured the traffic volume in KB/s, and all traces were kept in the text files. Finally, we processed all the files and calculate the traffic density in each link. The result will be presented in Section 6.3.

6.2.3 Call Setup Duration

Since the publish/subscribe paradigm has been proposed as a solution for the new Internet era; a comparison between the effectiveness of IP and pub/sub network, to handle the tasks in this thesis, is another interesting aspect to explore.

This test was performed by making a call from one client behind the Gateway_Right1 to the other client behind the Gateway_Left1, across both types of the networks. The packets were configured to travel over the same definite path at all time. The path, which had been used in this test, was the link number 1, 4, 6, 7, and 11. To achieve a fair comparison, we used the same Python code of our pub/sub gateway and SIP registrar for both cases. The code

was modified in order to communicate through the IP protocol instead of using the publish/subscribe paradigm, in the IP-based experiment. This ensures that the packets still pass through the same processes, and spend approximately the same time inside the gateway. Additionally, an IP address had been assigned to the SIP registrar at Forwarder5, and it was preconfigured at our IP gateways for user registration and call forwarding. This SIP registrar acted like a redirect server [29]. When the call was originated, it sent *302 Moved Temporarily,* with the callee's IP address, back to the caller so that the caller can directly reinitiate the call with that address.

To measure the call setup time, we divided the duration into 2 phases: Call Initiation and Call Acceptance. The call initiation phase is the period between the times, an INVITE message is sent out, and the subsequent 180 Ringing is received at the caller side. Likewise, the call acceptance phase takes place after the call is accepted, meaning that the 200 OK is sent to the call originator. The acceptance period lasts until the callee gets an acknowledgement. Finally, the media session is started.

We measured the duration from both phases at the positions shown in Figure 6.2. The call initiation phase was detected at the originating gateway, where an INVITE message was received from Client_Right (1), and the 180 Ringing was sent out from the Gateway_Right1 (2). The call acceptance was measured at the destination gateway, where the 200 OK was received from Client_Left (3), and the corresponding ACK was sent out of the Gateway_Left1 (4). The total call setup duration was the summation between the times from these two phases; and we performed the test ten times in each network.

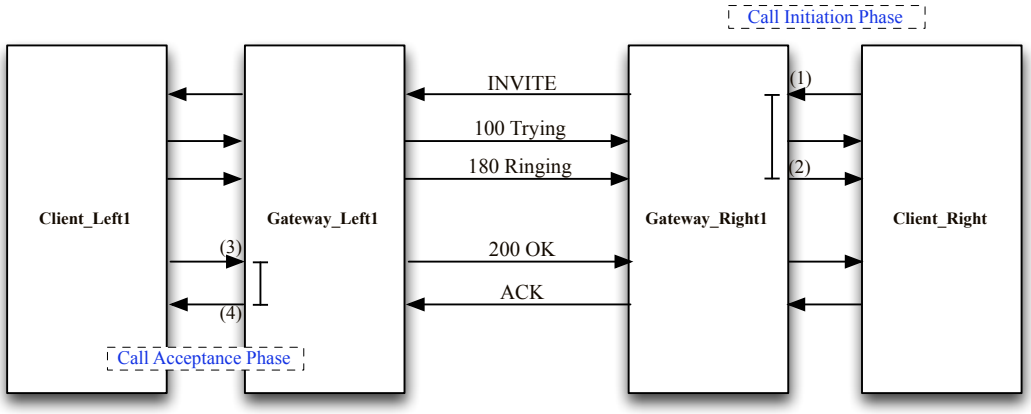


Figure 6.2: Figure illustrating points of measurement for call setup duration

6.3 Results and Analysis

After we had completed all the experiments in Section 6.2, we used Matlab [41] for data processing. We start from presenting the results and analysis of the functional testing in Section 6.3.1. Following this, we discuss the results from the traffic and call set-up measurements in Section 6.3.2 and 6.3.3 respectively.

6.3.1 Functional Testing

6.3.1.1 SIP Session

The analysis from Section 6.2.1 confirmed that our implementation supports the basic functionalities stated in Requirement 3.3.1. The call could be established in correspondence with the network conditions. The message display from the Blink application helped confirming that the gateway functionality was as sufficient as it was supposed to be.

Starting from the DNS Response, our gateway managed to retrieve the IP address of an interface, attaching to the IP network, and encapsulated it with the DNS response. Then, the IP host knew the IP address of the contacting gateway.

In the user registration process, our gateway was well synchronized with the SIP registrar; we could see that the registration and deregistration had taken place without any noticeable delay. Also, scopes and information items were published and subscribed in the pattern defined in Chapter 4. These items gave a clear data classification for each user and reduced the time for packet parsing. Moreover, the SIP registrar consumed quite little memory, as it stored only the necessary data. At last, when the user logged out of the domain, our pub/sub system unpublished and unsubscribed the user's related pieces of information, showing the flexibility in the design.

In SIP session initiation, if we compare our design to a redirect server and a proxy server for the IP network [29], our design combines the advantages from both of them. The caller did not need to wait for the *302 Moved Temporarily* like how things work in the redirect server. Moreover, the traffic did not have to pass the intermediate node all the time like the proxy server case. Instead, all the signaling and media traffic, except INVITE message, followed the direct optimal path provided by zFilter forwarding [44] and the Topology Manager [14]. Again, the decision to use Call-ID as a scope for each of the SIP sessions, guaranteed the uniqueness of the session, and made the signaling arrangement become simpler.

The experiment showed that all operations, supporting SIP voice call, worked in the same way as we have explained in our design. The conversations could be made among the user agents, including SIP conference. Furthermore, the call could be terminated by the UAS, via *BYE*, *603 Decline*, or *486 Busy Here*, and by the UAC via *CANCEL*. Call holding was also possible in this experiment. When the participant clicked the hold button, the media flowed in only one direction as stated in the SDP. Also, the session could be resumed, and the conversation was continued.

6.3.1.2 Multicast Session

To start with, the evaluation program showed that SAP messages traversed from the source to all other gateways in the network. Then, it could be forwarded further to the IP multicast address for SAP announcements as well. The packet was inspected, and we could make sure that the SAP message contained the right information related to the stream. By examining the state of the multicast session in our gateways, it was obvious that the gateway could select all relevant information from the SDP, and then created the pub/sub and IP mapping without any difficulty (Req. 3.3.3.1).

Meanwhile, the gateway was prepared to detect any kind of IGMP request. It added the client's IP address to the list, kept tracking number of clients, and managed to subscribe the stream, when necessary. That was when the first client joined the session. The video was displayed without distortion. In the other way round, the gateway unsubscribed the stream,

right after the last client left the session. These statements affirm the accuracy of the multicast session-management features (Req 3.3.3.2).

6.3.1.3 Reachability

Requirement 3.4.2 was satisfied by the result of this experiment. Media session could be established between all hosts regardless of their location. SIP users from all over the network could also make a voice communication between each other. The gateway consulted the registrar in order to find the location of the other party. If the destination of the call was not registered to the domain, our SIP registrar published a *404 Not Found* back to the caller, resulted in the call cancelation. For the case that the destination was in the same network, the gateway behaved like a SIP proxy server. It accepted the signal and media from the source and forwarded to the associated partners, through the Internet Protocol. Alternately, the gateway selected the publish/subscribe paradigm, when the destination was located in different IP networks.

6.3.1.4 Support for Concurrency

In this test, we had to recall that the objective of our implementation is to prove the concept of our design. Therefore, this scalability test was just to make sure that the gateway is functional by not taking into account other irrelevant issues. The gateway could support up to 15 concurrent SIP calls and multicast sessions in total. Hence, our design meets Requirement 3.4.3 as stated.

6.3.1.5 Adaptability

The adaptability test fulfilled Requirement 3.4.4 for both SIP session and multicast streaming. Regarding SIP, this could be noticed by the completeness of the SIP calls as well as new arrangement of the media ports.

For multicast, at the time Client1_1 firstly announced the multicast session, other gateways in the testbed parsed the SAP announcement correctly and created the multicast state using the original multicast address. Then, as soon as the Client2_1 streamed the media to the same multicast IP:port pair, we could see that the gateway noticed the abnormality by making a comparison between the information's unique identities, and immediately assigned another available multicast address to this new stream. The gateway handled this situation in the same manner for other following multicast sessions. Furthermore, the SAP packet was suitably edited in accordance with the mapping inside our gateway. This characteristic not only prevents an address overlapping, but also allows the client to precisely join the media without any mistake.

6.3.2 Traffic Monitoring

6.3.2.1 SIP Session

The graphs in Figure 6.3 are presenting the amount of traffic in each link during the SIP voice call. Link numbers on the x-axis are referred from the prototyping environment in Figure 6.1. The call was made from Client3 to Client1, which were the hosts sitting behind Gateway_Right1 and Gateway_Left1 respectively.

To begin with, all the graphs show that the traffic in pub/sub network is generated when an event occurs, and along the relevant path only. It can clearly be seen that the registration and deregistration phases produced the least amount of traffic load. This result conforms to our system design, since the SIP registration was accomplished within two message exchanges

between the gateway and SIP registrar, which are REGISTER and *200 OK*. In addition to the result from *ifstat*, our evaluation program showed that the registration was completed within approximately 100 ms for both clients.

Moving on to the signaling phase, the traffic occurred along the links 1, 4, 6, 7 and 11, due to the fact that there were three involving participants: Gateway_Left1, SIP registrar and Gateway_Right1. Besides, the traffic was increased to approximately 4.5 KB/s, because the SIP protocol requires five message exchanges in total (INVITE, *100 Trying*, *180 Ringing*, *200 OK*, and ACK), to establish the call. Also, it took two additional exchanges, between BYE and *200 OK*, to terminate the call.

The media session introduced the heaviest traffic load. The traffic periodically came from an RTP media session. The traffic density of the RTP and RTCP packets, during the conversation, varied within the range from 10 to 14 KB/s in this experiment. Ultimately, the traffic volume stayed quite the same until the call was terminated. Then, it turned back to an idle stage.

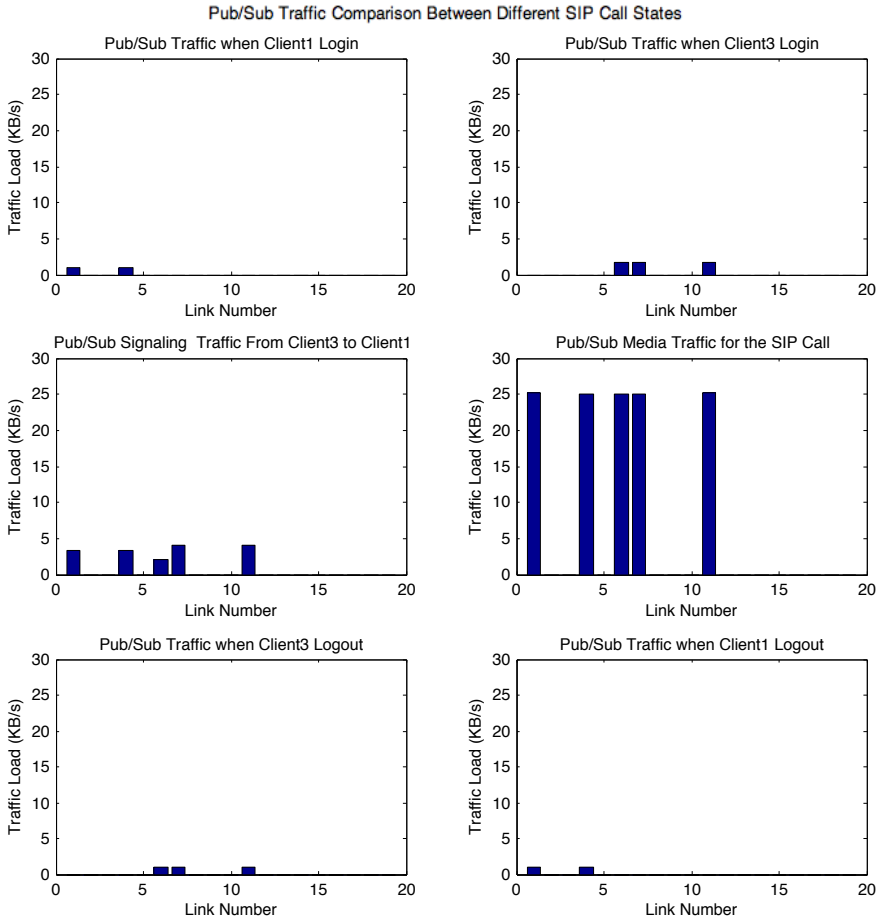


Figure 6.3: Pub/sub traffic from different SIP phases

6.3.2.2 Multicast Streaming Session

The traffic monitoring was significant for demonstrating the benefit of multicast. We averaged the traffic with respect to a ten-second window. From the packet trace, we could see that SAP announcement and RTCP sender report were transmitted twice during this period. On the contrary, the video packet was continuous and the speed varies from 40 to 250 KB/s. This condition depended on the video content and its resolution. In this section, we illustrate the result by referring to the prototyping architecture. They are shown in Figure 6.4 to 6.7. Different types of traffic are presented with different arrows, and the traffic density is demonstrated in terms of the line thickness. Alternatively, bar charts, summarizing the traffic load in IP and pub/sub networks during this experiment, are presented in Appendix B.

From Figure 6.4, although the media stream was available to the network, when there was no subscriber for this session, the traffic was limited to the source's IP subnet. This is an effect of the design as we chose to join the media right after the SAP is advertised. Also, it is a trade-off for the speed improvement, since we publish multicast items in advance. Only the SAP and RTCP packets could flow across the network, and it produced approximately 0.094 KB/s. In addition, it is noticeable that the packets were not flooded. Instead, they were forwarded to all destinations via optimal paths, resulted the traffic density to remain zero in link number 5, 10, 13, 15, 18, and 19. This observation is an expected outcome from the pub/sub Bloom filter forwarding. At this point, one noticeable advantage of the pub/sub gateway was to block redundant traffic from flowing into the global network.

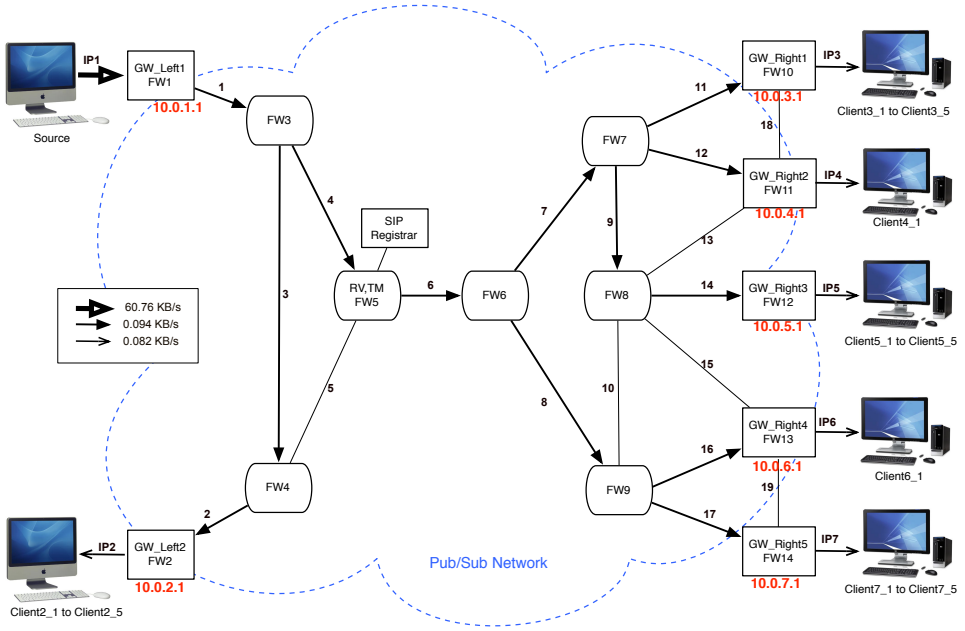


Figure 6.4: Pub/sub traffic for SAP announcement and RTCP sender report

After the first client behind Gateway_Right1 joined the media session, the stream traversed across the pub/sub network. The media flowed in the same manner as the SAP and RTCP messages by following the shortest path. Moreover, the amount of traffic was relatively the same in all links. This network state is shown by Figure 6.5.

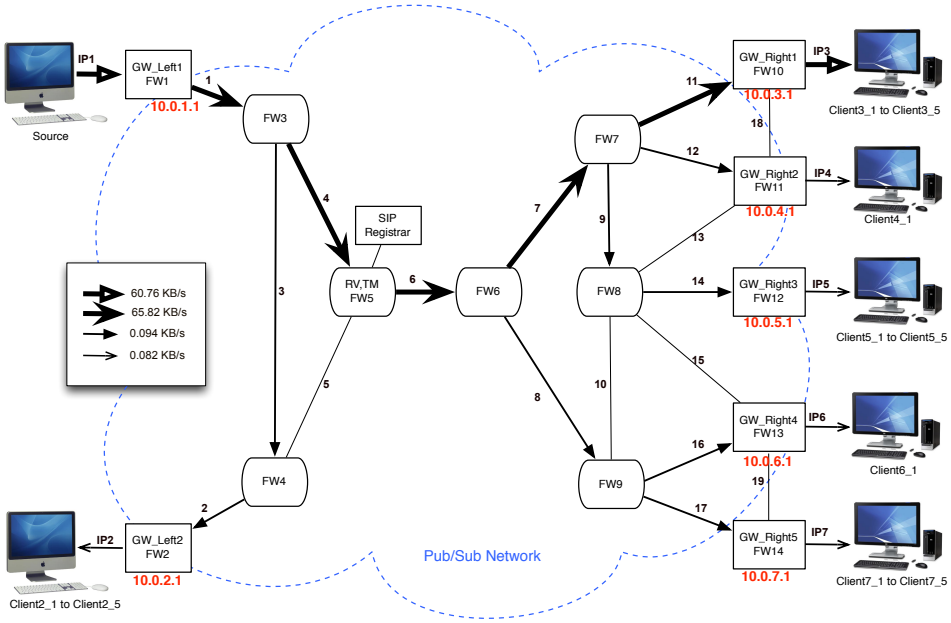


Figure 6.5: Multicast traffic after Client3_1 joined the session

The efficiency of the in-packet Bloom filter multicast was more apparent in the next experiment. After the clients behind Gateway_Left2 and Gateway_Right5 joined the session, Figure 6.6 shows that the media was multicast in the pub/sub network. We could see that regardless of the number of subscribers, the original traffic was maintained to be the same at all times. It certified that no packet duplication was needed to deliver the content to multiple destinations.

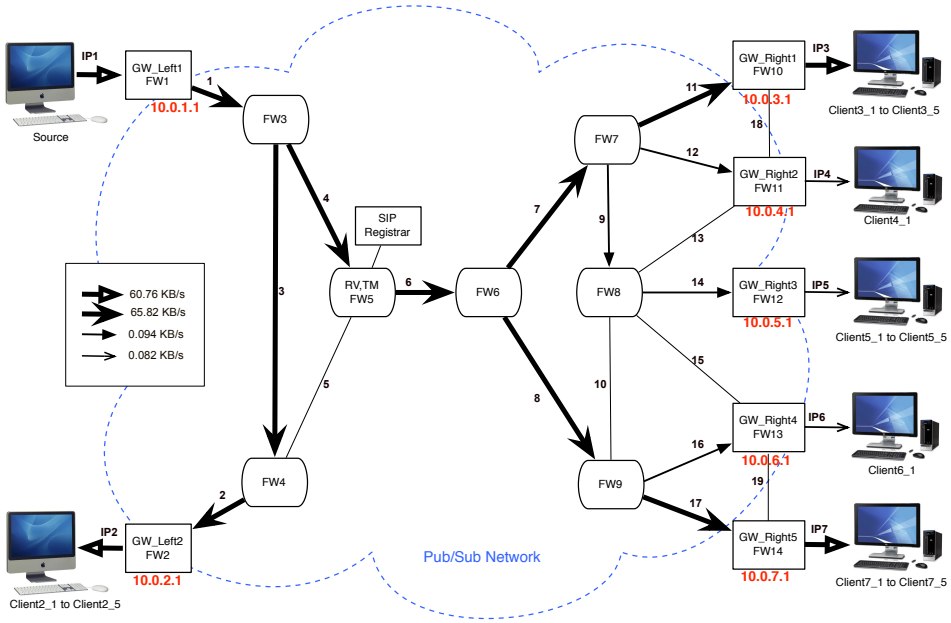


Figure 6.6: Multicast traffic after Client7_1 and Client2_1 joined the session

Figure 6.7 shows an example of the network traffic when the client has left the multicast session. Owing to the fact that our gateway is designed to track the amount of group members, the gateway unsubscribed the media from the pub/sub network only when the last client left the session. Then, the media flow towards that particular gateway was stopped. In other cases, the gateway kept streaming the media to the multicast address resulted the traffic level to be unchanged. Finally, the network condition turned back to the beginning phase merely with the SAP announcement and RTCP sender report, when there were no subscribers.

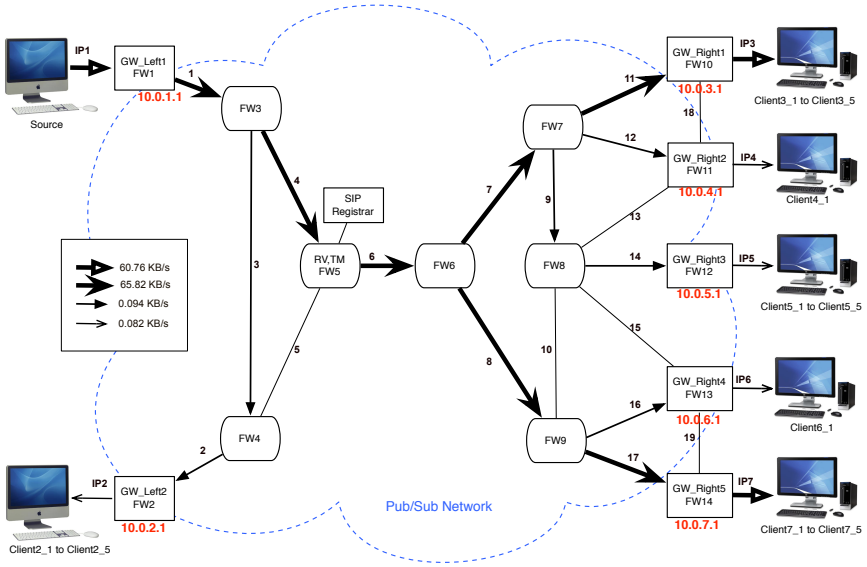


Figure 6.7: Multicast traffic volume after GW_Left2 unsubscribe the session

By comparing the traffic density between pub/sub and IP networks, the traffic was slightly higher in the pub/sub area. This can be explained by taking a look at the packet overheads. An IP and pub/sub format in Figure 6.8 to 6.9 show that the IP overhead is always fixed at 20 bytes, while the packet header in pub/sub varies according to the length of information identifier. In addition to this, our evaluation program confirmed that the pub/sub header of the SAP announcement was 30 bytes longer than the IP's.

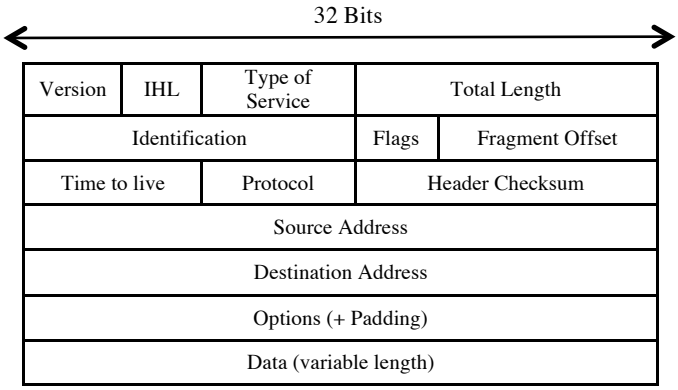


Figure 6.8: IP packet format

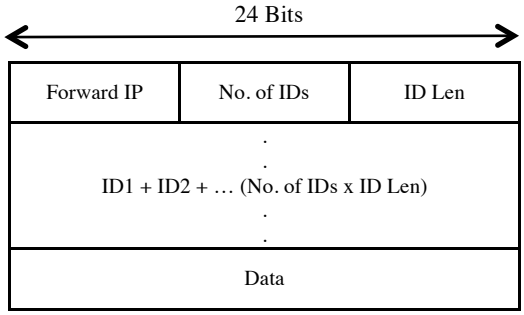


Figure 6.9: PURSUIT packet format for PUBLISH_DATA event

6.3.3 Call Setup Duration

In this section, two bar graphs were plotted according to the data from an individual test. The results from call initiation and call acceptance phases are presented separately in Figure 6.10 and 6.11. Incidentally, the accumulated call setup durations are presented in Figure 6.12.

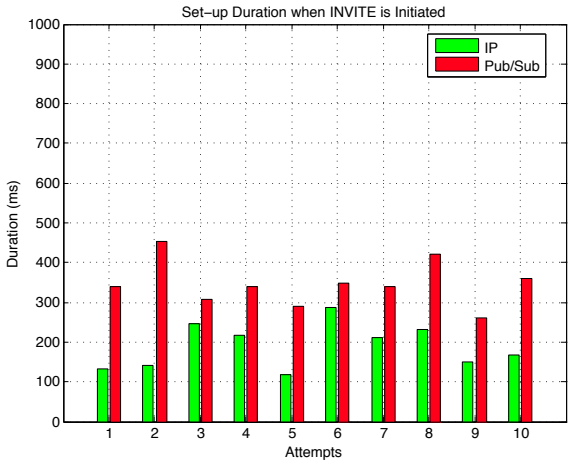


Figure 6.10: Call Setup duration from the Initiation phase

On the one hand, at the time when an INVITE message was originated, the result distinctly implies that the IP network took shorter time for the *180 Ringing* to arrive at the caller. This was consistent to the assumption we made before the experiment. The reason was because, the pub/sub gateway needed to take some time to publish and subscribe scopes, and information items before the actual data transfer. In contrast, the data was transferred right after it arrived at the IP gateway.

On the other hand, the result turned out to be the opposite after the callee had accepted the call. Once the information structure was ready, the pub/sub network seemed to provide a better speed for the data transfer as we could see that the call acceptance phase was shorter in the pub/sub network. This is a result from the underlying architecture, which is beyond the scope of this thesis. At present, there is a plan to make a comparison between the latency in IP and publish/subscribe networks in the PURSUIT project, which will clarify the reason for this situation. At this stage, there are two reasonable assumptions to explain this. First of all, the faster speed in the pub/sub network comes from the stateless Bloom filter forwarding

which consumes less time, comparing to the stateful routing mechanism in the IP protocol. Secondly, it is possible that our gateway spends more time to parse the IP packet. Even though we tried to strictly use the same amount of procedures in the experiment, the packet-level operation is still different. For example, in order to achieve the same piece of information, the gateway reads only pub/sub headers, while it has to parse the IP payload every time.

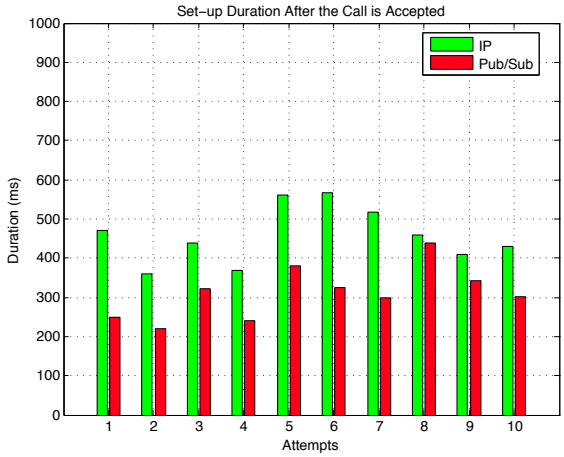


Figure 6.11: Call setup duration from the Acceptance phase

Subsequently, if we take a look at the accumulated call setup durations, the pub/sub mechanism gave a better call setup speed by 60% of all attempts. This time-based experiment is corresponding to the time, being used to transfer signaling traffic volume in Figure 6.3. We could see that the benefit of pub/sub transfer speed is not very apparent in a short-term communication. More precisely, when the packet is delivered once in a long while. Nonetheless, in a long-term communication, like multimedia session, the PURSUIT publish/subscribe architecture is considered to be a promising solution.

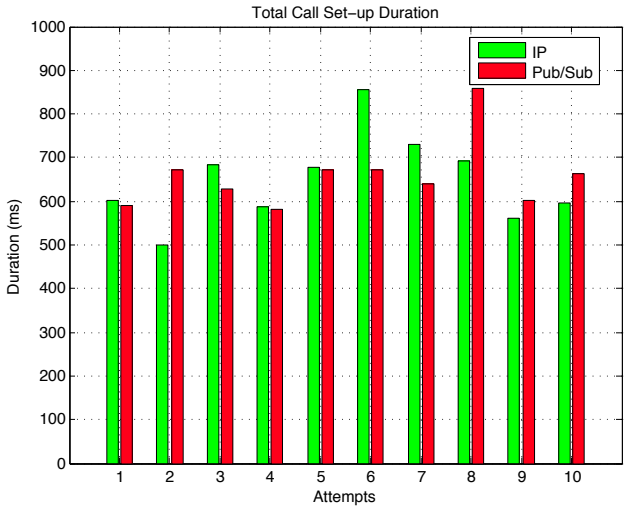


Figure 6.12: Total call setup duration

6.4 PURSUIT Publish/Subscribe Technology as the Internet Solution

In this section, we discuss benefits and drawbacks of the publish/subscribe network that we have observed during the thesis.

Real-time Communication over the Publish/Subscribe Technology

According to the observation during this thesis, PURSUIT publish/subscribe technology is compatible with real-time communication. It did not introduce any human perceivable delay. Likewise, voice and video were in a good quality.

Mobility

Mobility is one benefit of the PURSUIT technology. We could observe this quite obviously during the call setup evaluation in Section 6.2.3. When we were implementing an IP gateway, we noticed that the gateway needed to know an exact IP address of the SIP registrar to accomplish the user registration process. As a result, the IP address of SIP registrar has to be updated every time we change its location.

On the other hand, the knowledge above was unnecessary in the pub/sub network. The registrar can be situated in any pub/sub node with no additional configuration required. This property helps the service maintaining to become more convenient. For example, if a node with SIP registrar fails, we can seamlessly start the application on top of another pub/sub node without having to notify the gateway.

Pub/sub Header

The header length of pub/sub packets varies, depending on the length of information identifiers, and it is usually longer than the IP header. This causes the traffic density to be higher in most cases. On the other hand, there is an advantage out of this. The fact that the pub/sub header contains necessary metadata about the payload helps reducing the processing time and complexity. In the IP communication, we basically know only IP address and port. In order to know the type, purpose and meaning of each packet, we mainly need to parse the payload.

Chapter 7

Conclusions

In this thesis, we have proposed a novel design for a publish/subscribe gateway and SIP registrar, which enable IP-based real-time communication over the publish/subscribe paradigm. The design included the support for unicast and multicast traffic. First of all, we described how the gateway and pub/sub-based SIP registrar work to support voice communication over the Session Initiation Protocol. Secondly, we presented the gateway's mechanism to enable multimedia streaming over the multicast technique, as well as the system implementation.

The gateway's capability to convert IP to publish/subscribe traffic, and vice versa fulfils an objective to enable the co-existence between the IP and pub/sub networks. This is presented by the result of our design and implementation. Moreover, the result also confirmed that an idea to deploy the gateway during an IP to pub/sub migration is reasonable and applicable in practice.

The functional outcome of SIP voice call and multimedia streaming over multicast showed that the Session Initiation Protocol and multimedia streaming are possible to be implemented in an Information-Centric context. However, the designing phase introduced one noticeable challenge to the thesis. It is because the communication strategy within the Information-Centric Network is quite new, and works differently from how the Host-Centric system usually does. Therefore, we had to revise the design several times in order to achieve the one we proposed. Finally, we have also discovered both advantages and disadvantages of the publish/subscribe internetworking during this thesis.

One typical question for the future Internet proposal is whether the ICN is actually better than the recent IP network or not. In this thesis, we compared the publish/subscribe paradigm to IP networking in two different bases, i.e., traffic density and call setup duration. The traffic analysis showed that the pub/sub header varies upon the length of information identifier, while the size of the IP header is always 20 bytes. Despite the header length, we could see that the pub/sub header contains more information about the payload, which simplified the packet analysis in the implementation phase.

It seemed that within the scope of our evaluation, the speed of data transfer in the pub/sub prototype was slightly faster than the IP. Sixty percent of the pub/sub-based experiments provided faster call set-up duration. On the other hand, the pub/sub network needed not only some time to set up the information structure but also several signaling exchanges before the actual data can be transferred. This is obviously a tradeoff from the publish/subscribe networking. Nevertheless, the result from this thesis is not enough to make any conclusion for the question. Certainly, there should be more research, and we should take into account

other aspects; for example, global scale internetworking and more applications testing as well.

The scope of this thesis is a very fundamental step towards an actual deployment. The gateway can be developed further, and the ideas for future work are presented below.

Future Work

- **Scalability of the Design**

  The testbed in this thesis is not scalable enough to claim that this design is suitable for the global scale, e.g., up to millions gateways. Another reasonable step is to test and improve the information structure in our design to be scalable enough for supporting a very large amount of active sessions.

- **Globally Accessible Location Service**

  In this thesis, the location service is internally integrated with the SIP registrar. It can be further developed as another pub/sub application, which independently stores user information and provides a global access to other pub/sub entities as well as the registrar. This way, we can distribute the risk of failure, resulted the location service to remain functional if the SIP registrar happens to stop working. Additionally, we can restart the registrar anytime and at any node in the network, while the user information is still maintained.

- **More Use Cases for SIP Communication**

  Although, the Session Initiation Protocol is mostly used to establish a voice communication in practice, we can also use it to initiate other types of multimedia session, e.g., radio, video and gaming sessions. Therefore, the gateway can be developed to support these features over the publish/subscribe paradigm, as well as more complicated features of the voice call.

- **Support for Additional Protocols**

  It would be more profitable for the users if the publish/subscribe gateway supports additional protocols, enabling more applications to communicate over the publish/subscribe paradigm. Examples of popular applications are Web Browser with an HTTP [50] protocol, and e-mail services using SMTP [25], IMAP [36], and POP [26].
  In this case, the host are able to use their IP devices without any necessity to purchase a new one. Furthermore, the gateway can block redundant IP traffic, e.g., from data polling [28], and utilises the property of pub/sub so that the gateway can wait for a newly updated data without any further effort.

Bibliography

[1] Alan B. Johnson. SIP-Understanding the Session Initiation Protocol. Boston: Artech House.

[2] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV). RFC 2782, February 2000.

[3] A. Johnston, and O. Levin. Session Initiation Protocol (SIP) Call control – Conferencing for User Agents. RFC 4579, August 2006.

[4] AG Project, SIP Infrastructure Expert. http://www.ag-projects.com/projects-products-96/683-software-repositories. [Cited 5 September 2012].

[5] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol version3. RFC 3376, October 2002.

[6] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. Commun. ACM, 1970.

[7] Blink Site http://icanblink.com/. [Cited 15 October 2012]

[8] Esteve, P. Jokela, P. Nikander, M. Särelä, and J. Ylitalo. Self-routing Denial-of-Service Resistant Capabilities using In-packet Bloom Filters. EC2ND, pages 46-51, 2009.

[9] CCNx Project Main Site. http://www.ccnx.org [Cited 5 Sep 2012].

[10] Cohen, Bram (2001-07-02). "BitTorrent — a new P2P app". Yahoo eGroups. Retrieved 2007-04-15.

[11] D.D. Clark, K. Sollins, J. Wroclawski, and T. Faber. Addressing Reality: An Architectural Response to Real-World Demands on the Evolving Internet. *ACM SIGCOMM Computer Communication Review*, August 2003.

[12] D. Mayer. Administratively Scoped IP Multicast. RFC 2365, July 1998.

[13] D. Trossen (ed). Conceptual Architecture: Principles, Patterns and Subcomponents descriptions. PURSUIT Deliverable D2.2, May 2011.

[14] D. Trossen (ed). Architecture Definition, Components Descriptions and Requirements. PURSUIT Deliverable D2.3, October 2011.

[15] Facebook Main Site. http://www.facebook.com [Cited 3 July 2012].

[16] Future Internet Assembly Site. http://www.future-internet.eu/home/future-internet-assembly.html [Cited 18 October 2012].

[17] G. Camrillo, J. Ott, and K. Drage. The Binary Floor Control Protocol (BFCP). RFC 4582, November 2006.

[18] H. Schulzrinne. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 1890, January 1996.

[19] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[20] H. Schulzrinne and S. Casner. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551, July 2003.

[21] Ifstat Site. http://gael.roualland.free.fr/ifstat/. [Cited 5 September 2012].

[22] Internet Assigned Numbers Authority (IANA) Official Site. http://www.iana.org/. [Cited 21 October 2012]

[23] J. Kjällman (ed). First Lifecycle Prototype Implementation. PURSUIT Deliverable D3.2, September 2011.

[24] J. Kjällman (ed). Progress Report of Component Implementation. PURSUIT Deliverable D3.3, March 2012.

[25] J. Klensin (ed). Simple Mail Transfer Protocol. RFC 2821, April 2001.

[26] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939, May 1996.

[27] J. Postel. User Datagram Protocol. RFC 768, August 1980.

[28] J. P. Martin-Flatin. Push vs. Pull in Web-Based Network Management. IM'99, May 1999.

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

[30] J. Rosenburg and H. Schulzrinn. An Offer/Answer Model with the Session Description Protocol (SDP). RFC 3264, June 2002.

[31] J. Rosenberg. A Framework for Conferencing with the Session Initiation Protocol (SIP). RFC 4353, February 2006.

[32] J. Rosenberg, H. Schulzrinne, and O. Levin (ed.). A Session Initiation Protocol (SIP) Event Package for Conference State. RFC 4575, August 2006.

[33] LinkedIn Main Site. http://www.linkedin.com [Cited 3 July 2012].

[34] M. Cotton, L. Vegoda, and D. Meyer. IANA Guidelines for IPv4 Multicast Address Assignments. RFC 5771, March 2010.

[35] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335, August 2011.

[36] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501, March 2003.

[37] M. Handley, C. Perkins and E. Whelan. Session Announcement Protocol (SAP). RFC 2974, October 2000.

[38] M. Handley, V. Jacobson, C. Perkins. 2006. SDP: Session Description Protocol. RFC 4566, July 2006.

[39] M. Mealling and R.Daniel. The Naming Authority Pointer (NAPTR) DNS Resource Record. RFC 2815, September 2000.

[40] M. Särelä, C. Esteve, T. Aura, A. Zahemszky, P. Nikander, and J. Ott. Forwarding Anomalies in Bloom Filter Based Multicast. IEEE INFOCOM, 2010.

[41] Matlab Site. http://www.mathworks.se/products/matlab/.[Cited 5 September 2012].

[42] Orebaugh, Angela; Ramirez, Gilbert; Beale, Jay (February 14, 2007). *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress. p. 448. ISBN 1-59749-073-3

[43] P. Eugster, P. Felber, R. Guerraqui, and A. Kermarrec. The many faces of publish/subscribe. ACM Computing Surveys, June 2003.

[44] P. Jokela et al. LIPSIN: Line Speed Publish/Subscribe Inter-Networking. ACM SIGCOMM, August 2009.

[45] P. Mockapetris. Domain Names – Concepts and Facilities. RFC 1034, November 1987.

[46] P. Mockapetris. Domain Names – Implementation and Specification. RFC 1035, November 1987.

[47] P. Savola. Overview of the Internet Multicast Routing Architecture. RFC 5110, January 2008.

[48] PSIRP Project Main Site. http://www.psirp.org [Cited 1 July 2012].

[49] Python Programming Language. http://www.python.org [Cited 27 August 2012].

[50] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616, June 1999.

[51] R. M. Magalhaes. Session Initiation Protocol (SIP) and Its Functions. http://www.windowsnetworking.com/articles_tutorials/Session-Initiation-Protocol-Functions.html, February 2005 [Cited 24 July 2012].

[52] S. Romano and M. Stahl. Internet Numbers. RFC 1020, November 1987.

[53] S. Ruthfield. The Internet's History and Development From Wartime Tool to the Fish-Cam. *Crossroads* 2.1, September 1995.

[54] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. DNS Extensions to Support IP Version 6. RFC 3596, October 2003.

[55] Scalable and Adaptive Internet Solutions (SAIL) Project Main Site. http://www.sail-project.eu [Cited 5 September 2012].

[56] Seventh Framework Programme. http://cordis.europa.eu/fp7/home_en.html [Cited 1 July 2012].

[57] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, August 1998.

[58] The Click Modular Router Project. http://www.read.cs.ucla.edu/click/click [Cited 30 August 2012].

[59] VideoLan Organization, VLC Media Player for Ubuntu. http://www.videolan.org/vlc/download-ubuntu.html. [Cited 5 September 2012].

[60] VLC Site http://www.videolan.org/. [Cited 15 October 2012]

Appendix A

SIP Headers and SDP Attributes

Table A. 1: Frequently used SIP Header Fields according to RFC 3261

Header Fields	**Meaning/Purpose**
Accept:	Specifies acceptable media field
Allow:	List of allowed methods at the end host
Call-ID:	A unique identifier of each media session
Call-Info:	Additional knowledge about the call
Contact:	Contains URI according to the method specific purpose
Content-Length:	Tells the size of message body
Content-Type:	Provides information of the message body format
CSeq:	Sequence number of the specific message type and method
From:	Contains URI of the request originator
Max-Forwards:	Maximum number of forwarding hops
Require:	Contains what user agent wants another SIP entities to support
Route:	Used for controlling the route of the particular request
To:	Contains User URI of the receiver
User-Agent:	Tells information about the user agent Ex: Application name
Via:	Keeps tracking the entity along the path

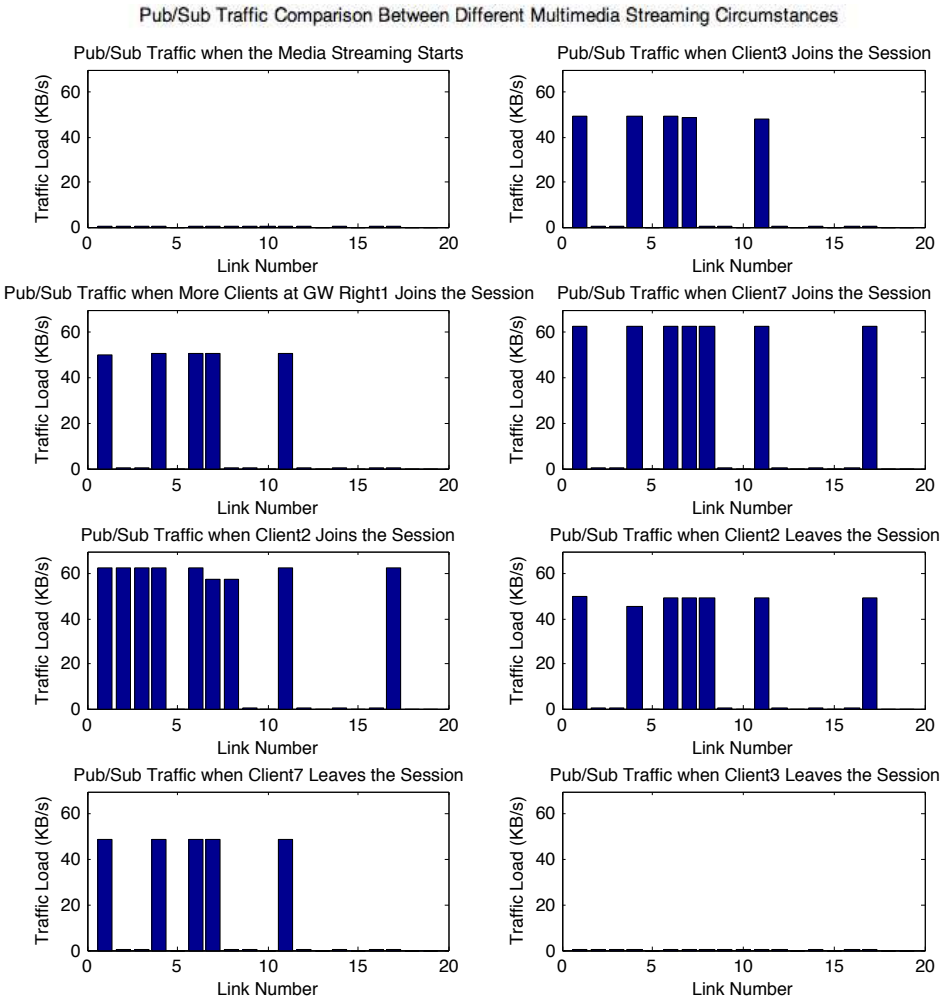
Table A. 2: SDP Attributes from RFC 4566

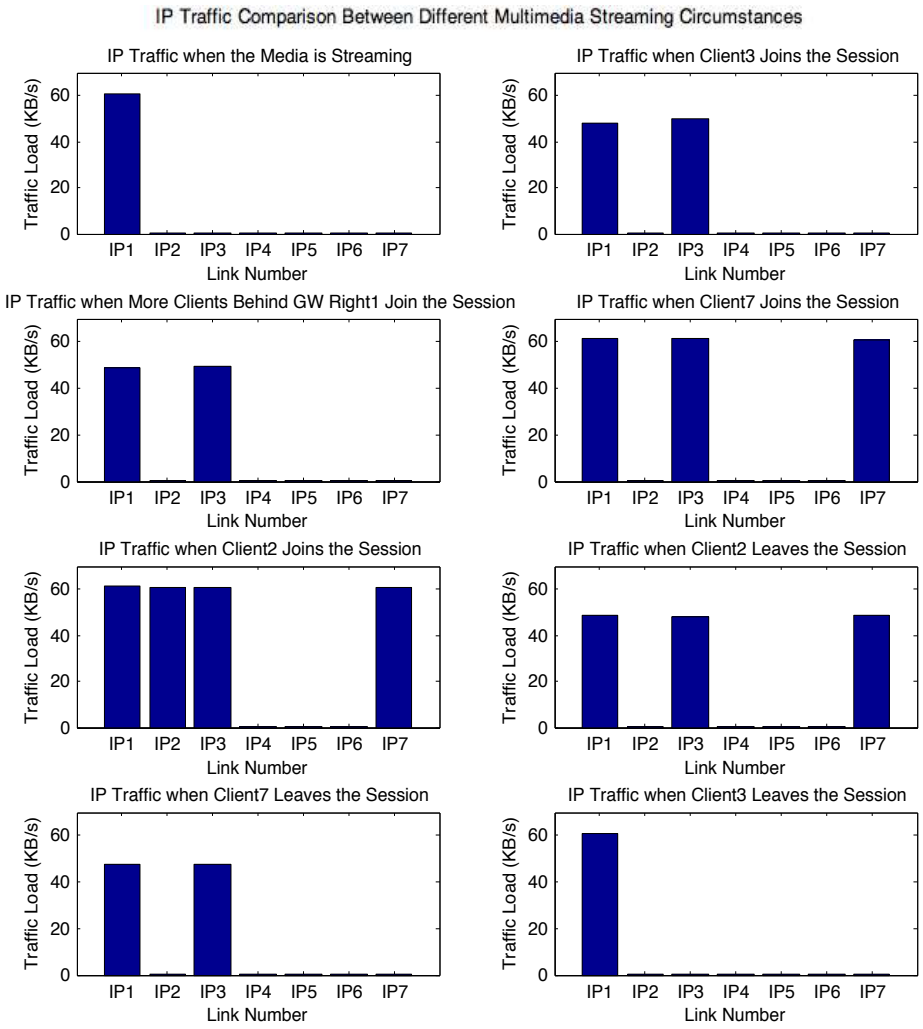
SDP Attributes	**Meaning**	**Mandatory**
Session Description		
v=	SDP version	✓
o=	Owner of the session	✓
s=	Session name	✓
i=	Session information	-
u=	URI of the description	-
e=	E-mail address	-
p=	Phone number	-
c=	Connection information	-
b=	Bandwidth	-
z=	Time zone	-
k=	Encryption key	-
a=	Session Attribute	-
Time Description		
t=	Starting and ending time of the session	✓
r=	Repeat time	-
Media Description		
m=	Media name and transport address	✓
i=	Media title	-
c=	Connection information	-
b=	Bandwidth	-
k=	Encryption key	-
a=	Session Attribute	-

Appendix B

Multicast Traffic in Pub/Sub Network

The following graphs present the average traffic volume from different multicasting circumstances. The x-axis represents the link number according to the Figure 6.1, while the y-axis gives the information about the average traffic load.




IP Traffic Comparison Between Different Multimedia Streaming Circumstances
IP Traffic when the Media is Streaming
IP Traffic when Client3 Joins the Session
IP Traffic when More Clients Behind GW Right1 Join the Session
IP Traffic when Client7 Joins the Session
IP Traffic when Client2 Joins the Session
IP Traffic when Client2 Leaves the Session
IP Traffic when Client7 Leaves the Session
IP Traffic when Client3 Leaves the Session
Traffic Load (KB/s)
0
20
40
60
IP1 IP2 IP3 IP4 IP5 IP6 IP7
Link Number