Tampere University

Trung Van

INTERNET-OF-THINGS STREAMING OVER REALTIME TRANSPORT PROTOCOL

A reusablility-oriented approach to enable IoT Streaming

Master of Science
Faculty of Engineering and Natural Sciences
Examiners: Prof. Jose Luis Martinez Lastra
University Instructor. Luis Enrique Gonzalez Moctezuma
October 2022

CONTENTS

ABSTRACT

Trung Van: Internet-of-Things Streaming over Realtime Transport Protocol
Master of Science
Tampere University
Factory Automation and Robotics
October 2022

The Internet of Things (IoT) as a group of technologies is gaining momentum to become a prominent factor for novel applications. The existence of high computing capability and the vast amount of IoT devices can be observed in the market today. However, transport protocols are also required to bridge these two advantages.

This thesis discussed the delivery of IoT through the lens of a few selected streaming protocols, which are Realtime Transport Protocol(RTP) and its cooperatives like RTP Control Protocol(RTCP) and Session Initiation Protocol (SIP). These protocols support multimedia content transfer with a heavy-stream characteristic requirement.

The main contribution of this work was the multi-layer reusability schema for IoT streaming over RTP. IoT streaming as a new concept was defined, and its characteristics were introduced to clarify its requirements. After that, the RTP stacks and their commercial implementation-VoLTE(Voice over LTE) were investigated to collect technical insights. Based on this distilled knowledge, the application areas for IoT usage and the adopting methods were described.

In addition to the realization, prototypes were made to be a proof of concept for streaming IoT data with RTP functionalities on distanced devices. These prototypes proved the possibility of applying the same duo-plane architect (signaling/data transferring) widely used in RTP implementation for multimedia services. Following a standard IETF, this implementation is a minimal example of adopting an existing standard for IoT streaming applications.

Keywords: IoT, Streaming, real-time, RTP, RTCP, SIP, SDP, VoLTE, eMTC, 5G, reusability, standardization, pub/sub

PREFACE

The past two years of my master's studies in Factory Automation and Robotics at Tampere Universities have been a great journey. This thesis is my final project, and I started it with some knowledge of the IoT domain and some telecommunication experience. Despite this, I decided to challenge myself by working on something novel that could impact people positively. It turned out to be enabling IoT streaming using the existing audio-visual streaming technologies. Throughout this process, I learned many valuable scientific research lessons and practical implementation techniques, all part of a fantastic learning experience. I genuinely enjoyed this journey.

To this thesis' readers, the document aimed to create content that an audience with different technical backgrounds could understand. To ensure the information is accessible, I separated the more technical sections from those focused on applications. This enabled readers to choose which section they would like to read first and in what order. I hope readers will gain insight into their chosen areas while learning something new from this document, regardless of their current knowledge base or expertise level. My aim was for you all to enjoy your reading experience no matter where you start!

I am incredibly grateful to my supervisor from Ericsson, Christer Holmberg, for his dedicated work and guidance throughout completing my master's thesis. His expertise in research and development was invaluable as I explored the many possibilities within the Internet of Things area. He provided meaningful insights that enabled me to complete my thesis effectively. In addition to working with Christer Holmberg, I had a chance to interact with other colleagues on the IoT Research team at Ericsson. They were all very welcoming and eager to share their knowledge about various topics related to this field of study. This experience taught me how important it is for people from different backgrounds to come together to succeed as a team, which I will carry into future projects or endeavors. Special thanks go out to Tampere Universities, including professor Jose Lastra who supervised my master's thesis, alongside Luis, who was also instrumental in helping ensure everything went smoothly during this process.

Finally, I thank my family and friends for being there throughout this journey.

Espoo, 18th October 2022

Trung Van

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Background

In recent years of IoT evolution, the data generated by IoT applications and the available Audio Visual (A/V) streaming technologies have formed a strong association. From the IoT perspective, stream liked data is becoming more ubiquitous for some classes of devices. This rising can be seen in the adoption of new applications that require time-series data like healthcare[1], smart agriculture [2], or robotics [3] to name only a few. Regarding the device number, the existing and incoming IoT applications will add up to 14.7 billion M2M connections in 2023 based on Cisco Annual Report [4]. In addition, technology-enabling factors discovered in different domains, like computing, data storage, and data transmission, can support the streaming process of IoT data. To summarize, regarding this high temporal resolution IoT data, there exist:

- The demand to invest in this topic.
- The numerous sources to collect data from.
- Technologies to collect, store and process this kind of data.

Approaching from the opposite direction, one can also observe the potential of applying what has been invented/invested in multimedia streaming for IoT. 82% of all consumer Internet traffic is for video traffic according to Cisco [5], and audio/visual content consumption can be easily found everywhere within the IP network coverage. These facts prove that a significant investment has been placed into the network infrastructure and the software/protocol inventions for multimedia streaming. Reusing this investment is in the interest of multiple parties, for example, the Internet Service Providers, Network Service Providers, and Network End Users. IoT developers also benefit from the reusing strategy thanks to a short roll-out time materializing their ideas into actual products. In a word, multimedia streaming offers IoT streaming reusability in different aspects like infra, software, protocols, regulatory procedures, and potential customers...

The motivation for this work was based firmly on the explained association. This thesis attempted to investigate what is provided by multimedia streaming that can satisfy the requirements of IoT streaming. Among many protocols involved in the multimedia streaming world, Realtime Transport Protocol (RTP) and the others related to its operation were

chosen to be in focus.

1.2 Problem statement

The thesis aimed to answer the following questions:

> `Question 1` How can the RTP built-in features and extensions be utilized for IoT Streaming?
>
> `Question 2` How the A/V infrastructure can be reused for IoT Streaming?
>
> `Question 3` How the A/V Conference Services can be used to realize an IoT Pub-Sub Service?

`Question 1` focused on the technical details, hence required deep-diving analysis on the functionalities of the chosen protocols. Each RTP functionality was described with real-world use cases. `Question 2` provided insights into the feasibility and benefits of multiple parties if A/V streaming infrastructure can be used for IoT streaming. As PubSub is a widely-used and powerful paradigm in the field of IoT, `Question 3` was added as a solution to provide PubSub-liked behaviors for a system given the available functionalities from the A/V Conference Services.

As can be seen from these questions, the priority of this thesis was to enable IoT Streaming using the same technology and infrastructure as A/V streaming. Hence, to better narrow down the context of this thesis and reduce confusion, the following questions or claims below were considered out of scope:

- Whether RTP can be used to transport non-A/V data?
- Is RTP a better fit for IoT Streaming compared with other IoT dedicated protocols?

1.3 Thesis structure

This thesis followed a rigorous structure to ensure a comprehensive and insightful examination of the research questions. The structure was designed to provide a clear and logical progression of the research, from the initial rationale to the conclusions.

Section 1 set the foundation for the thesis by presenting the research questions and their significance in the context of the IoT streaming landscape. This section established the need for the research and the study's objectives, providing a road map for the rest of the thesis.

Section 2 was devoted to the related work, the involved specifications, and the literature review, which served two critical purposes. Firstly, it allowed the author to position their work within the broader academic discourse, highlighting the thesis's contribution. Secondly, it gave the reader an understanding of the fundamental concepts and technical

details of the chosen technologies, laying the groundwork for the subsequent sections.

Section 3 was dedicated to demonstrating the IoT use cases for the selected technologies. This section illustrated how the technologies could be applied in various IoT scenarios, highlighting their versatility and potential.

Section 4 presented the prototypes developed to showcase the practical realization of the technologies. This section demonstrated the feasibility and effectiveness of the proposed approach and provided a tangible illustration of the thesis's contribution.

Section 5 analyzed the results of the prototypes and provided a comprehensive discussion of the findings.

Finally, in Section 6, the thesis's achievements were summarized, and possible future work was identified. This section concluded the thesis by providing a comprehensive overview of the study's contributions and implications for future research.

Overall, the structure of the thesis provided a robust and comprehensive framework for the research, allowing the author to thoroughly investigate the research questions and present their findings clearly and concisely.

2. TECHNOLOGIES OVERVIEW AND ANALYSIS

2.1 Literature review

In this section, some works from other researchers were mentioned to provide an overview of the research area and clustered into different groups in the *Related work* to support the contribution claims in the following *Summary*.

2.1.1 Related work

In recent years, IoT has generated many exciting topics in its fusions with multimedia and streaming technology.

Internet of Multimedia Things (IoMT), as defined in [6], is a subclass of IoT that requires more resources (due to faster processing speed, larger memory, and higher bandwidth consumption) to meet the additional requirements of Quality of Service (QoS) and Quality of Experience (QoE). Regarding the audio context of IoMT, IoAuT (Internet of Audio Things) was coined to conveniently discuss the interaction and cooperation between *Audio Things* [7] over the Internet. The same research group also introduced the Internet of Musical Things(IoMusT)[8] as a subclass of IoT dedicated to serving music-related purposes. In [9], IoT and a telephony service called Voice over IP have been integrated to achieve multiple goals, like introducing a more straightforward user interface or supporting applications involving human-human interaction. The group of authors referred to this paradigm as VIoT. Regarding stream processing for IoT, the discussion is usually about: architectural decisions, as depicted in "The Lambda and the Kappa" [10] or real-time processors/frameworks comparison on the application level like between Apache Spark, Flink, and Storm [11].

There are also some intersecting works of IoT and RTP/RTCP protocol. For example, the survey in [12] provided a comprehensive overview of various streaming protocols (including RTP/RTCP) for IoT data transmission. In [13], the authors delivered two contributions: using additional fields in the RTP/RTCP header to provide information about the IoT environment and dividing a multimedia session into smaller sessions. In addition, they also proposed an adaptive transmission control mechanism based on the information from the newly defined headers to avoid stream flooding problems. The experiments in [13] and

[14] showed that the IoT variant of RTP outperformed the standard version for multimedia transmission. Three session layer protocols: MQTT-SN, CoAP, and RTP, have been compared in [15] for audio, speech, and video transmission. From the experimental analysis, CoAP and RTP both demonstrated the lowest application packet loss. However, the RTP/SIP stack was claimed to be less supported in IoT firmware than the others. SIP has also been investigated as an essential protocol in the signaling plane of many RTP implementations. From work in CoSIP-[16] and Lightweight SIP-[17], the group of authors demonstrated the possibility of reducing the packet size while securing the reusability of SIP application-layer libraries.

2.1.2 Summary

From the related work, it can be seen that the synergy between IoT and multimedia streaming has been investigated using different approaches. One was to start with multimedia-enabled devices and then form a network using such devices with connectivity enabled. This approach can be seen in IoMT and IoMusT. Similarly, protocols like IoT-RTP/IoT-RTCP, while operating under a constrained environment, were still used to transmit only multimedia data in the mentioned experiments. Alternatively, an existing IoT application can be upgraded by adding a media function, as in VIoT. Another approach would be introducing an IoT-adapted and interoperable version for current tech, as done for CoSIP. In addition, these researches were concentrated on bench-marking while real-world applications were usually out of scope.

These observations led to two ideas for this thesis contribution. First, by expanding the usage of RTP over conventional A/V streaming, there was a significant potential to utilize RTP to deliver IoT data (not just device condition) and take advantage of the protocol's existing functionalities. Second, to support multimedia streaming infra re-purposing, there should be an application-focused and illustrative study to connect the commercial RTP implementing systems and various innovative use cases coming out from the IoT world.

2.2 Essential concepts

This section provides the essential concepts of IoT and Stream that served as elementary ideas for the following discussions.

2.2.1 Stream

In this document, a **stream** is used to refer to a sequence of events with the following properties:

Unbounded : The number of events occurring within a period is theoretically unbounded. There

could be zero to an infinite number of events within a period. However, due to the device's limitations, an RTP stream has a limit of sampled events. The unboundedness, however, is helpful to emphasize that a stream should be able to transmit a large number of events.

Time-serial : Each event is associated with a time value. The unitary period between two consecutive time values in a stream defines the temporal resolution of that stream. An application considered real-time if events happen within a unitary period doesn't have any actionable consequence on the application. The requirement for this property depends on the application. For example, if the temperature of a room is less likely to fluctuate in the magnitude of 10℃ within 0.1 seconds, any application that can update the temperature more often than 0.1 seconds can be considered real-time (for that specific use case). The time-serial property also requires a linear and monotonic time value to be the reference.

Unidirectional The direction of a stream is only from a source to a destination. Two endpoints can have bidirectional streaming communication, requiring two separate streams.

2.2.2 Internet of Things - IoT

In the Y2060 Recommendation [18] from the International Telecommunication Union, **IoT** was defined as "A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies". A **Thing** in the IoT is an object of the physical/information world with capabilities of identification within and integration into the communication networks. Due to its massive number and diversity of devices, IoT intersects with multiple technology concepts: Machine-to-Machine(M2M) or Machine-to-People(M2P) in communication, Wireless Sensor Networks (WSN) in connectivity or Big Data in data management and analytics. In addition, while constraint devices are often thought of for IoT applications, IoT devices are not necessarily resource-limited. Therefore, in this thesis work, IoT is used with its broad definition as in [18], which does not narrow the scope of this term to only constraint devices.

2.2.3 IoT Streaming

In this thesis, IoT Stream is defined as a sequence of packets containing IoT data as described in 2.2.4, with the characteristics defined in 2.2.1. IoT Streaming involves a set of operations related to an IoT Stream.

2.2.4 Data involved in IoT Streaming

As the approach of this thesis is oriented toward re-using multimedia service infra for IoT purposes, the term A/V data and IoT data is used frequently, so they are defined in this sub-section for better clarity. A/V data includes audio and video that have been used ubiquitously and are well-standardized. Some examples of A/V data standardization could be in H.264 format for video or Opus format for audio. Because these data types are mainly used for human consumption, they can also be referred to as media, conventional media, multimedia, or human-consuming data. IoT data is usually an umbrella term for any data transmitted between IoT devices, resulting in a broad definition. This thesis uses the term IoT data for any type other than conventional audiovisual data. Some examples of IoT data can be:

- The light intensity measurements of photo-receptor in lumen(lm).
- The commands to instruct the autonomous vehicles.
- The multiplexed telemetry values (altitude, received signal power, current power...) from a drone.
- The multiplexed values from both video signal and the operational telemetry of a robot arm (joint configuration, tools orientation, motor speed...).

Some examples of data that is not IoT data defined above are:

- Video streams from a surveillance system.
- Music streams from an entertainment platform.
- A/V streams being multicasted in a conference application.

2.2.5 AV vs Non-AV

A large section of the current streaming use cases is to support AV content delivery, as discussed in the Background Section. To reuse the same infrastructure for non-AV data, it is helpful to locate some differences between them. AV data is designed to deliver content for human consumers. Hence QoS alone can not be sufficient for the quality assessment. AV content would require, in addition to QoS, the QoE and potentially QoP (Quality of Perception)[19] as investigated in [20]. Unlike AV data, Non-A/V data can not be assessed by human perceiving factors because the end user is often not human. The effort has been put from both industrial (ITU Technical Specification D4.4 [21]) and research[22] areas to specify the data quality management for IoT/non-AV data. Time-to-content is another factor that differentiates between AV and non-AV. Human user experience is strongly affected by the time between user interactions and the delivered content, as insight from this announcement[23]. A sufficient time-to-content and a reliable latency during the usage time are required for AV data. Because machines are the primary users of non-AV data,

latency is less crucial for some devices. However, reducing latency benefits other demanding applications, like Time Critical Communication. In addition, AV content is likely to be bulkier because AV-enabled end devices like smartphones or laptops are already sufficient for resource-consuming tasks.

2.2.6 Non-streaming vs Streaming

In this thesis, Streaming refers to a data distribution method that was not Request/Response based, and implied the following properties:

- The being interested changes are constantly updated with no query made in advance.
- The Streaming-based setup and configuration for transmission are expected to be less frequent than the Non-streaming.
- As packets are sent more frequently, the size of each Streaming packet is expected to be smaller than one from the Request/Response model, which may require data aggregation or contain a bulkier header.
- Streaming model is capable of providing higher time resolution data if needed.

2.3 RTP/RTCP

RTP is the network protocol to deliver audio and video content over IP networks. In addition, VoLTE networks employ RTP as the de facto protocol on their data-transferring plane. Therefore, it is necessary to provide some technical background before further reusability analysis. This section demonstrates RTP and its controlling complement protocol RTCP. Several central concepts of an RTP/RTCP are also introduced to provide the context for further discussion. The third subsection is dedicated to some currently standardized and popular functionalities from the IETF for RTP/RTCP. In the last subsection, some drawbacks of the RTP/RTCP stack are listed and discussed.

2.3.1 RTP

RTP is a transport protocol on top of IP/UDP designed for time-sensitive services specified by IETF(Internet Engineering Task Force) in RFC3550. The implemented system is expected to be robust on top of an unreliable network. From the design perspective, RTP protocol follows two main philosophies of *application-level framing* and *end-to-end*.

The first philosophy is based on the premise that the application using the transported data should be the entity to decide how to transport those data. In this case, an application can perform different behaviors such as retransmission, unchecked continuation

with less-than-perfect data, or sending new data just for fixing purposes. Consequently, solutions like a strictly defined transport protocol or retransmission by default protocol are insufficient for the aforementioned behaviors. Therefore, the application layer and the transport layer must cooperate closely with the aim that the transport layer processes the transmitted data while preserving boundaries from the Application Data Unit (ADU), and the application is exposed to the transmission error to react upon.

By applying the second philosophy, the intelligence is shifted towards the endpoints to form a system composed of network-aware end devices and dumb networks. Because the endpoints offload some work from the network, its resources focus on networking tasks and enable more participants.

An RTP packet consists of a fixed header, a list of contributing sources that could be empty, and a payload. The format of an RTP header frame can be seen in Figure 2.1 below.

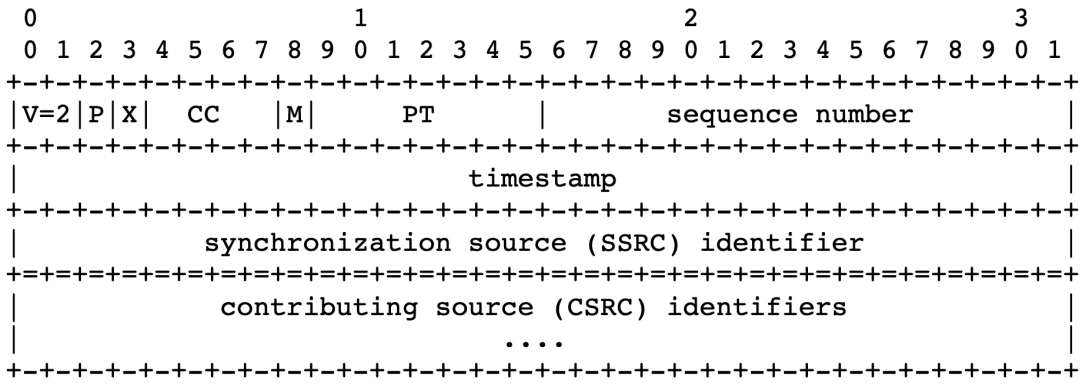

Figure 2.1. *RTP fixed header. Adapted from [24]. One row is 32 bits*

Among the illustrated fields, only the *CSRC* - the list of identifiers for the sources contributing to the delivered payload - is optional. The first three rows in Figure 2.1 must be the beginning of any valid RTP packet. The compulsory fields and their meanings can be summarized as:

V *version* of the current RTP is 2 (1-bit).

P *padding* indicates if padding was applied on the packet to maintain a fixed size for all packets (1-bit).

X *extension* indicates if there is an extension for the RTP header applied on this packet (1-bit).

CC *CSRC count* holds the number of CSRC identifiers if there are in the packet (4-bits).

M *marker* has different interpretations depending on the packet profile and is usually used to mark the packet corresponding to a significant event (1-bit).

PR *payload type* indicates the format of this RTP payload via the agreed RTP profile (7-bit).

sequence number is an increasing monotonic sequence corresponding to the being sent RTP packets to support packet loss detection (16-bit).

timestamp marks the sampling instant of the data in the payload (32-bit).

SSRC *source ID* is used to classify packets from different sources for synchronization.

Thanks to the fixed fields of an RTP header, several features can be considered evident. The data type of RTP packets during a session is deterministic and can be either static for some predefined types or dynamically assigned in the case of more recent types. Thanks to the timestamp, packet reordering, and delivery monitoring can be implemented, although RTP does not provide any QoS or guaranteed delivery as out-of-the-box features. RTP also inherits supported features from the chosen underlying protocol, like multicast. Devices can utilize the SSRC fields to differentiate packets from multiple sources in the de-multiplex phase. If it requires additional payload-format-independent fields to support the desired function, those fields can be included in the header extension right after the default fixed header with the X-field set to one. The additional fields must fit into only one extension whose format can be seen in Figure 2.2. If there exists an RTP implementation that does not support a specific extension, the RFC3550 specified such implementation to ignore the extension header to maintain the interoperability of the system.

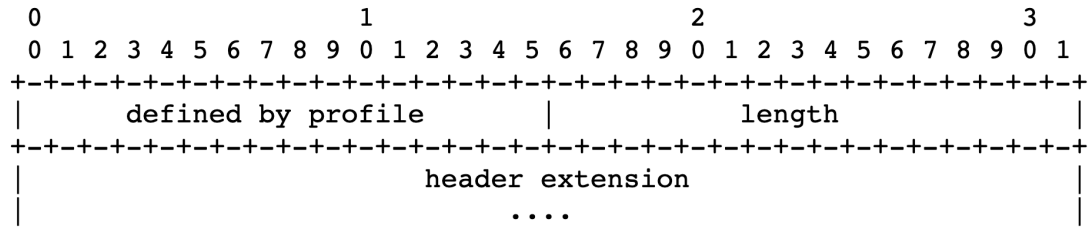

Figure 2.2. *RTP extension header format. Adapted from [24]. One row is 32 bits*

In an RTP deploying system, a participant is an entity that is reachable via a signaling address. Accordingly, an RTP session, a central concept of RTP, is defined as a group of participants interacting with each other via RTP. An RTP session can be dynamically configured as unicast or multicast. A participant is allowed to engage in multiple sessions at the same time.

2.3.2 RTCP

To provide control and monitoring for RTP operation in the data transfer plane, RTCP is often implemented jointly within the same system. From [24], the protocol on which RTP/RTCP will operate must support multiplexing control and data packets to implement RTCP. The control functions of RTCP can be listed as follows:

- Providing data distribution feedback to senders via receivers' reports which can be used for flow and congestion control.
- Providing a transport-layer identifier for streams of the same source called canonical name (CNAME). This ID is also more persistent than the SSRC, which could be altered in case of ID conflicts or program reset.
- Providing a global overview of the RTP session as the control information is distributed to all participants.
- Providing optional and minimal information for session control like email, participation ID, and location.

RTCP packets are exchanged periodically in comparison with the event-based characteristic of RTP. The format of an RTCP packet is illustrated in Figure 2.3

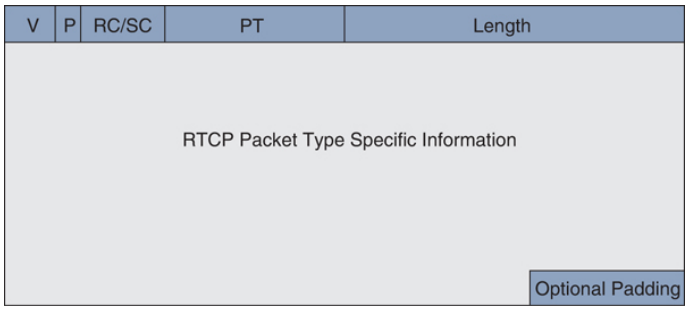

Figure 2.3. *Common RTCP packet format. Adapted from [25]*

Among the common fields, the V and P fields (the first 3 bits) gain the exact definition as in an RTP header. The following 5 bits are used to either indicate the number of received reports (RC) included in a packet or the number of SSRC/CSRC chunks if the packet is of type SDES. The field PT is used to indicate the predefined types of component packets. The last 16 bits denote the payload length of an RTCP compound packet. The control information conveyed via RTCP includes:

SR (PT=200) Sender Report contains the statistics from the active sender(s).

RR (PT=201) Receiver Report contains the statistics non-active sender(s).

SDES (PT=202) Source Description describes the source of being sent payload.

BYE (PT=203) indicates the end of participation from a participant.

APP (PT=204) could contain various types of payload depending on specific applications.

A sent RTCP packet must always be compound, i.e., an RTCP packet must be composed of at least two individual packets listed above. The first component packet must be an SR or an RR packet to facilitate their RTCP header for the compound packet.

Despite the exact distribution mechanism, the two protocols use two distinguished types

of packets to achieve function modularization, simplicity, and efficiency. The modularization allows the RTP/RTCP to support the loosely-coupled control for multicast use cases. Control-related information could be accessed without exposing the actual data and, on top of that, could be shared among participants or dedicated controlling third-party entities. The demultiplex functionality is passed on to the transportation protocol to maintain simplicity. In addition, an implementation also performs more efficiently if similar processing steps are not scattered over multiple layers [26], which was also mentioned as *Integrated Layer Processing* in Clark and Tennenhouse's work [27].

2.3.3 Functionalities

2.3.3.1 Multicasting

The need to send identical data to multiple participants is ubiquitous in communication networks and is usually called group communication. The transmission within this type of communication could be one-to-many or many-to-many. While using many unicast streams could also provide, to a degree, the function of group communication, it can be seen that the unicast solution consumes more resources like network bandwidth and device memory compared with multicast. In RTP, the multicast distribution is only provided if the underlying network supports it. In this thesis work, RTP multicast resembles the RFC IP multicast definition [28]: transmitting RTP packets to a group of interested participants or a multicast group.

Currently, there are two models of multicast in RTP: *Any Source Multicast (ASM)* and *Source Specific Multicast - (SSM)*. The terminologies to distinguish those two models can be found in Table 2.1. In ASM, each participant only specifies its multicast group; hence traffic from one participant in a group is always distributed to the rest of that group. While ASM provides the many-to-many by default, it consumes more resources and is highly complex to maintain such a multicast function. Furthermore, while some applications demand many-to-many communication, others may not and waste a large part of the mesh connection. For that reason, in cases where the transmission is one-to-many and predominantly unidirectional on the data transfer plane, SSM is a more efficient solution. If SSM is applied, each participant specifies a source in addition to the multicast group. Accordingly, that participant only receives traffic from the specified source address.

Service Model:	ASM	SSM
Multicast Abstraction:	group	channel
Identifier:	G	S, G
Receiver Operations:	Join, Leave	Subscribe, Unsubscribe

Table 2.1. *ASM vs SSM terminologies*

For a clearer explanation, some illustrative terminologies are needed for describing SSM:

Distribution Source is the only component that distributes RTP packets to the receivers.

Original Sender is an endpoint that originates RTP packets in a session. This definition is similar to a source in RFC3550 or the Media Sender in RFC5760 [29]. Still, this term is needed to explicitly separate this source from Distribution Source and generalize the data type of RTP packets.

Feedback Target is the logical component to whose address the RTCP reports from receivers are directed. This feedback information is unicast. A receiver can be associated with more than one Feedback Target.

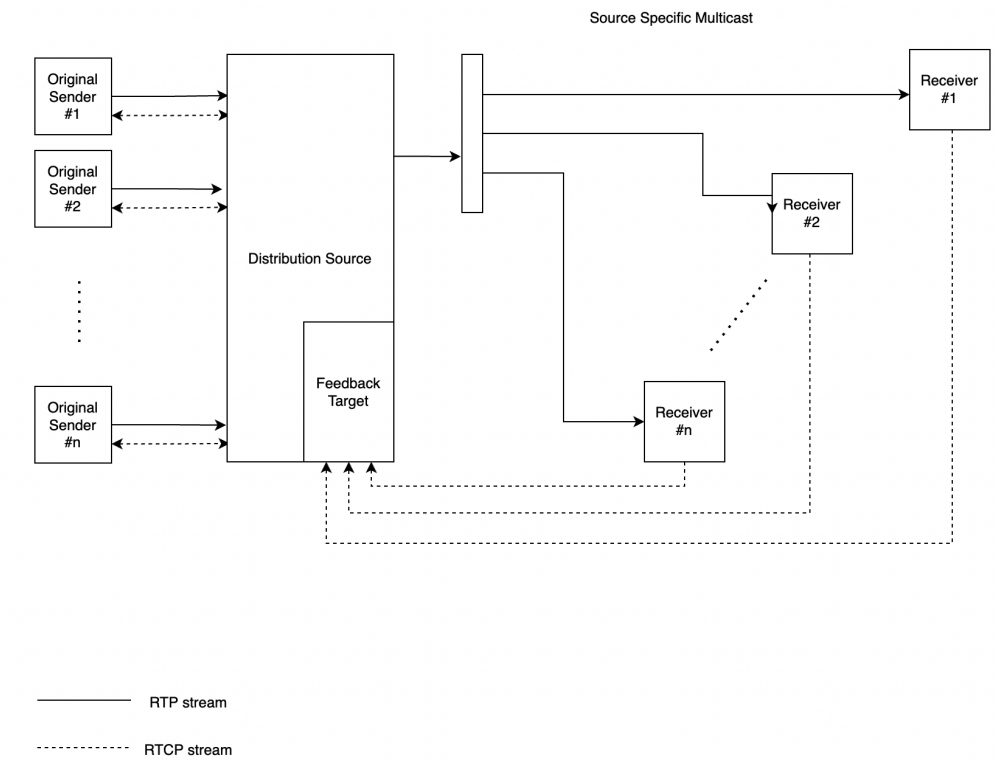

Figure 2.4. *SSM system architecture with integrated Feedback Target*

The Feedback Target(s) can be integrated into the same entity with the Distribution Source (Figure 2.4), but this architecture is not compulsory. There are various topologies for

organizing Feedback Targets, such as disjoint architectures: parallel, hierarchical, or in a chain, but the coherent RTCP information sharing between the Feedback Targets and the Distribution Source must be satisfied as an essential requirement. Figure 2.5 illustrates an example system architecture of an RTP implementation with disjoint Feedback Target instances connected in parallel to the Distribution Source. To suit various topologies using unicast RTCP feedback, a system could use the "Simple Feedback Model" or the "Distribution Source Feedback Summary Model" as mentioned in Section 6, and Section 7 of [29]. Generally, the first option proposes all RTCP packets arrived at a Feedback Target are kept intact and relayed to the Distribution Source while in the second option, the Feedback Target extracts the information from RTCP packets and sends only the summary version to the Distribution Source.

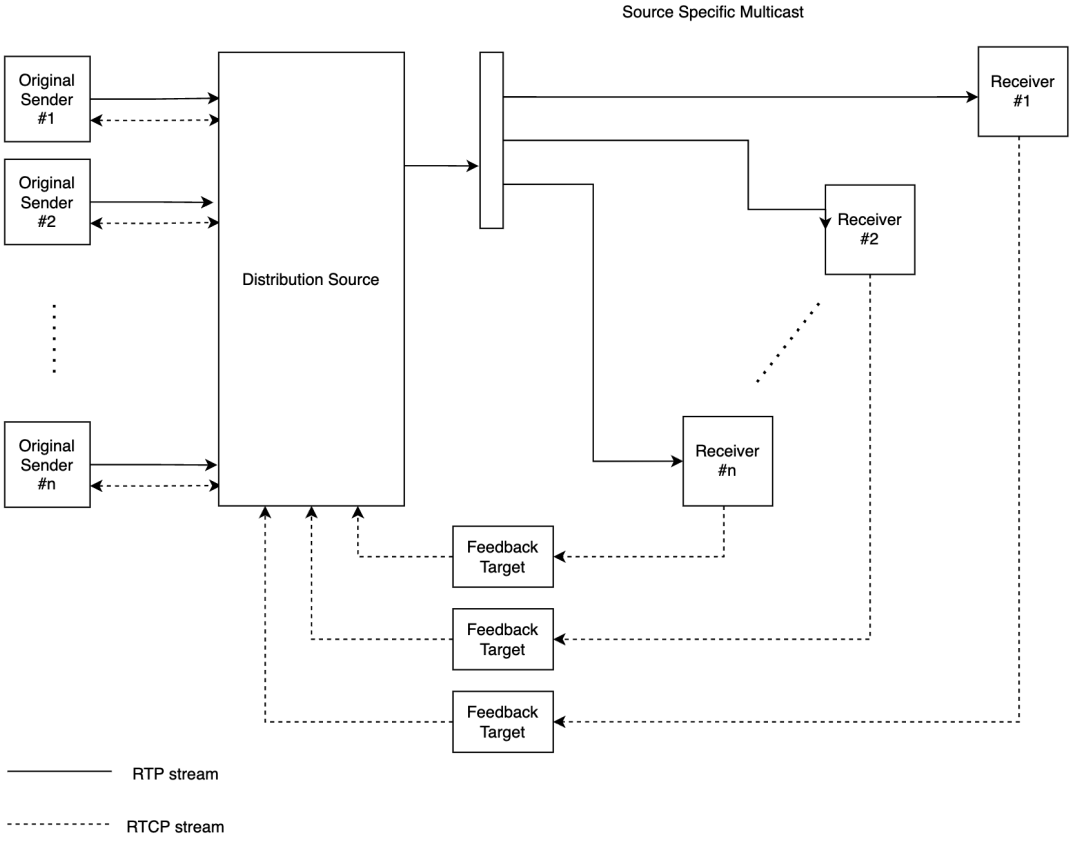

Figure 2.5. *SSM system architecture with disjoint and parallel Feedback Targets*

2.3.3.2 Multiplexing

Multiplexing is an operation in which data from multiple inputs (packets from different sources) are aggregated onto some common medium to be transmitted. The multiplexed packets share temporal(receiving at nearly the same time) and spatial(receiving at the same address:port) relationships. In this thesis work, RTP multiplexing means transmit-

ting multiple RTP streams using a single 5-tuple. Because packets from various streams are received at the same address and port, a reversing operation, so-called demultiplexing, of packet sorting based on their original streams must be done

An endpoint may consider supporting multiplexing for different motivations, and three examples could be:

- A single physical source supports various streams of varying data types, and one wants to preserve that physical binding during the transmission phase
- A source with simulcast capability is set to transmit multiple resolution data into one port
- Re-transmitted, or Forward Error Correction packets are sent using the same means as the original packets.

Each individual among the multiplexed entities(packets) must be addressable, which means it can be separated after the multiplexing and transmission process. Figure 2.6 illustrates a simple example of multiplex-enabled RTP implementation. In that example, packets from multiple sources are gathered, arranged, and sent at a multiplexing point. On the other end of the stream, a demultiplexing point operates the opposite function to classify and forward packets based on their sources.

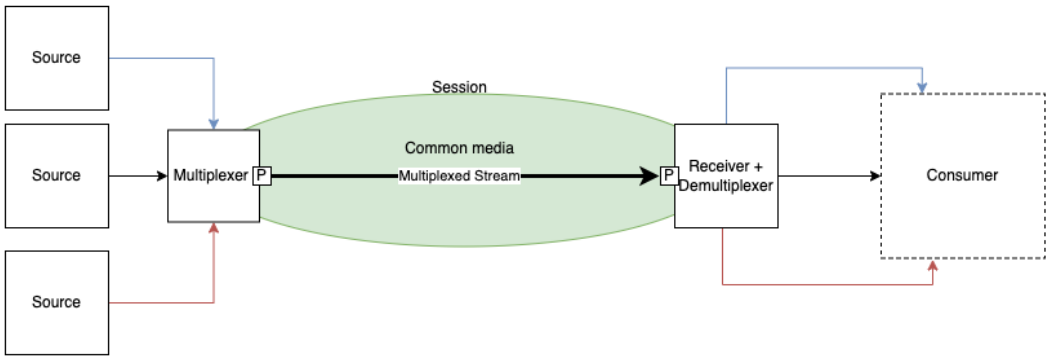

Figure 2.6. *An example diagram of multiplex*

Since its establishment, RTP has been updated with many extensions to enhance this grouping behavior, and as a summary, RFC8872 [30] divided the available binding solutions into categories as below:

- By signaling means:
  - RTP/RTCP based: using the built-in IDs of RTCP like CNAME in SDES.
  - Out-of-band signaling based: using the identifiers and descriptions in the signaling plane, for example, with SDP.

- By binding scope:
  - Binding streams within an RTP session using SSRCs or SSRC-multiplexing.
  - Binding streams across multiple RTP sessions or Session-multiplexing

For SSRC-multiplexing, each source produces an RTP stream identified by the SSRC field. The multiplexing function is already agreed upon in advance during the session negotiation. The sender sends multiple RTP streams as a single multiplexed stream in an RTP session using the same 5-tuple value. On the receiver's side, packets are sorted into different RTP streams using the SSRC field in each RTP header.

The Session-multiplexing slightly differs from the RTP multiplexing definition at the beginning of this section. Related RTP streams are transmitted in different RTP sessions using different values of ports and addresses. Consequently, Session-multiplexing preserves only the temporal relationship of the multiplexed streams. Session-multiplexing requires an additional mechanism to identify sessions and bind multiplexed sessions together.

Depending on the specific requirements for a system deploying RTP, one can decide to implement RTP functionalities following one of the designs in Table 2.7

Name	Design	Pros	Cons
Multiple Data Types One Session (MDT-OS)	Using a single RTP session. Each stream carries packets of one data type.	• Minimal resource in terms of traversal cost, state recording, overhead operations. • Allocation and configuration are on RTP level.	• Share-fate streams in the same session. • Incapable to provide finer feedback, control and management at stream-level. • Risk of SSRC collisions in case an application interacts with multiple sessions. • May require SSRC record and translation. • Bandwidth negotiation only supports SDP. • A session could be underused if its application requires only a small fraction of the total streams. • Interoperability concern for legacy implementations.
One Data Type One Session (ODT-OS) using multiple SSRCs	A single data type in a session. As the number of data types is limited, so is the number of sessions. There could be multiple streams in one session from a single or multiple endpoints.	• By design suitable for applications that require multiple data types separation. • Flow-based QoS can be tailor for different data types. • Low overhead for the security association. • Dynamic streams that rely on session state can take advantage of the shared information instead of additional signaling.	• More sessions need to be managed. • Interoperability concern for legacy implementations. • Security association is available only at session level.
One Data Type Multiple Sessions (ODT-MS)	A session contains streams of a single data type. Number of single-data-type sessions can be scaled depending on the needs from an application.	• Horizontally scalability. • Supports finer stream separation. • Less need for SSRC-specific signaling. • Enables flow-based QoS prioritization. • Key-management mechanisms at this scope are available to be used.	• More sessions need to be managed. • Multiple sessions binding is required. • Interoperability concern for legacy implementations. • Security association overhead is higher. • Lack of participant-level control.
One Endpoint as One SSRC (OE-OS)	Each endpoint in a point-to-point is assigned to a single SSRC which results in a two-SSRC session. The corresponding session could be either unidirectional or bidirectional.	• Great support for legacy interoperability. • Specific requirements can be negotiated for each endpoint pair using SDP description in the signaling plan. • Supports security association mechanisms at session level.	• High complexity due to large number of sessions. • High resource demands for example to maintain the correct state records or number of used ports. • Higher delay. • Significant overhead in case of dynamic or frequently configured sessions (quick entering/leaving participants). • The same SSRC is required to be reused for multiple sessions while supposed to be randomly generated. • May require SSRCs record and translation.

Figure 2.7. *Multiplex designs overview*

2.3.3.3 Redundancy

Because RTP is developed on top of UDP, it is by design unreliable. An intuitive solution is to send the additional versions of that very packet. This approach is called redundancy. Regarding the packet content, the redundancy-based solution could use duplicated or non-identical redundant packets. The redundancy principle is built upon the small probability that both the original packet and the redundant could be lost during transmission. Sending redundant packets requires an opposite process of classifying the received packets to maintain the uniqueness of each packet is called "deduplication" or "streams merging".

In RFC7198 [31], Begen and Perkins defined the elementary concepts for RTP redundancy and suggested a few relevant considerations for implementation. The discussed solution in this document is passive and duplication based. The other solutions are demonstrated in later sections of this thesis. An important definition is *Dual Streaming*

which refers to "a technique that involves transmitting two redundant RTP streams (the original plus the duplicated) of the same content". Using either of the streams is sufficient for the receiver to playback in case there is no packet loss in the selected stream. It was also mentioned in RFC7198 that increasing the number of redundant streams to more than two would result in an out-weighted overhead over the protection benefits. Two RTP streams are evaluated to be equal in the routing plane if the IP addresses of their sources and destination are equal.

There are currently two distinguished types of RTP redundancy, namely *Temporal Redundancy* and *Spatial Redundancy*. In both cases, the RTP duplicated packet contains the same timestamp and sequence number as the original; hence the receivers could operate the deduplication using these identical fields. More details information, like the association between the original and redundant RTP stream or the RTCP operation for redundancy, can be found in Appendix A.1

Temporal redundancy

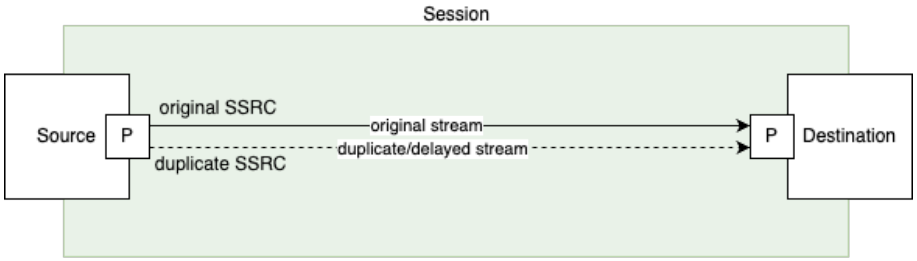

Figure 2.8. *Temporal redundancy visualization*

The first variation is designed to cope with transient loss, i.e., the loss is abrupt, sudden, and not likely to happen again on the same transmission path. Operating within the same session (Figure 2.8), the duplicated streams utilize the same path (same source/destination address:port pairs) as the original stream. However, their SSRCs are different to provide minimal source identification and thus conform to the requirement specified in [24]. Accordingly, the SDP content must include the attribute line `"a=ssrc-group:DUP"` followed by the corresponding SSRCs and another attribute `"a=duplication-delay:<value>"` was also added to support a receiver to allocate resource for its buffer. Temporal redundancy, as described above, can also be viewed as overlapping a delayed version of the original over the same session and stream configuration (except the SSRC).

Spatial redundancy

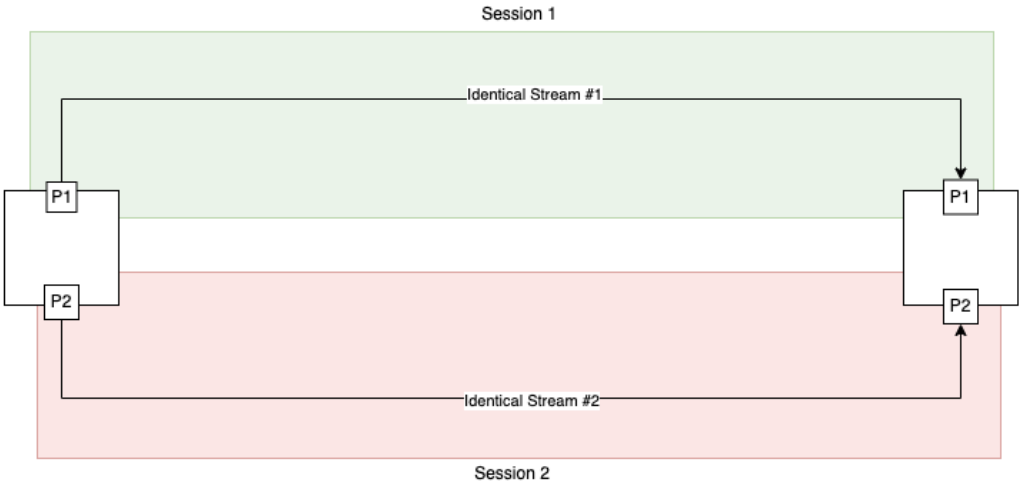

Figure 2.9. *Spatial redundancy visualization for inter-session scenario*

The second variation of redundancy is a solution aiming to solve a more persisting packet loss. The duplicate and original streams are sent from the source and destination using different pairs of address:port, indicating redundancy in several streaming paths. As a result, those two streams are allocated in different sessions. The SSRC values of the original and duplicate source, randomly generated for each, are hardly identical. This then allows one to use either `"a=group:DUP"` or `"a=ssrc-group:DUP"` to specify the original-duplicate association between steams. The former option is not available in temporal redundancy and officially stated in RFC7104[32] as SDP Duplication Grouping Semantics.

To summarize, both variations provide bounded delay and deterministic maximum bandwidth consumption as the redundant transmission is negotiated in advance. The number of participants experiencing the packet loss does not affect the recovering behavior, and no retransmission signal is required to implement redundancy. To be noticed, redundancy is not applicable for scenarios where the leading cause of packet loss is congestion because more redundant packets will stress the network even more. The recommended use cases for RTP redundancy are temporary link/network outages, given a precondition that the redundant traffic is within the network's capability.

2.3.3.4 Simulcasting

Simulcast is a blending of two terms, "simultaneous" and "broadcast," and got popularized by broadcasting services like television. In telecommunication, simulcast is defined as sending different representations of the same content over the same medium simultaneously. RFC8853 [33] defines simulcast function support for RTP. The particularly focused topology consists of three components:

- A source can generate streams of different encoding schemes.

- An intermediary component receives different encoded streams. It then forms a suitable set of streams for each participant and forwards the selected set of streams to the participants.
- A participant receives the forwarded streams and has a specific requirement for that stream.

The second component from the list corresponds to the RTP mixer specified in RFC7667. Two approaches are mentioned in RFC8853 to implement stream adaptive forwarding, which can be used for simulcasting: Transcoding and Switching.

Transcoding

This approach requires sufficient resources from the mixer as multiple demanding operations are assigned for a mixer, which must be able to:

- Decode the received streams.
- Encode the most optimal stream from the decoded streams for each and all of its receivers (participants).
- Forward the encoded streams to each participant.

While transcoding provides the highest adaptive ability for each receiver, some challenges/drawbacks exist to using Transcoding. The decoding and re-decoding are computationally expensive with a significant additional delay. A mixer is expected to keep track of the requirement of each receiver for the correct output format and the format of each received(input) stream. Another drawback is the risk of exposing transmitted content at those intermediary points in the network.

Switching

In the Switching approach, there is no intermediate decoding-encoding in the mixer. Instead, a forwarded stream is formed using a subset of the received streams. A less complicated operation results in a minor end-to-end delay, less pervasive content access, and resource demands on a mixer. These benefits come with the price of less adaptive mixing and higher network consumption.

In the signal plane, the simulcast function is described using SDP with an additional attribute `"a=simulcast"`. This new attribute also provides the number of simulcast streams plus their potential alternatives in each direction (`"send"` or `"receive"`). The chosen formats of these streams are restricted using the restriction identifier (`"a=rid"`) attribute of SDP defined in the RFC8851[34].

The scope of RFC8853 is limited to simulcast streams within one RTP session on top of one transport flow. However, the RTP taxonomies in [RFC7656] [35] suggest that more than one transport flow could be involved in a simulcast operation. Future work can extend

the function of simulcast to multiple RTP sessions/transport flows and aim for better QoS monitoring and robustness of a streaming application.

2.3.3.5 Pause and Resume

In many real-time applications, there are scenarios even when data packets are not actively sent between participants, the whole setting for communication is still wished to persist. This persistent setting allows participants to continue their transmission immediately. In streaming terms, a system must be able to *pause* and *resume* its streams to acquire such behavior. Pause-resume functions are ubiquitous, especially in recent multimedia applications, for example: a user pauses the being streamed video later on resumes that video with a modified setting, a participant in a conference temporarily removes his video from the presentation or a mixer entity pauses a subset of its simulcast streams in response to changes in network resources.

RFC7288 [36] describes the related aspects of an RTP/RTCP implementation to acquire pause-resume functionalities. To signal pause-resume behaviors, an implementation utilizes Feedback Messages (FM) which conforms to the Codec Control Message (CCM) specified in RFC5104 [37]. The defined Feedback Messages are listed in Table 2.2. These FMs are called blocks of Feedback Control Information and can form an AVPF RTCP feedback message as defined in RFC4585-"Extended RTP Profile for RTCP-Based Feedback" [38]. Thanks to this composability, a single message can carry many control information for one or more senders.

Message Type (RFC7728)	FM Category (RFC5104)	Function description
PAUSE	Request	Describes a request from a receiver to a sender to pause a stream
PAUSED	Indication	Sent by a sender to a receiver to indicate whether a stream is paused
RESUME	Request	Describes a request from a receiver to to a sender to resume a stream
REFUSED	Notification	Sent by a sender to a receiver to indicate a PAUSE or RESUME is not to be processed

Table 2.2. *Pause-resume Feedback Messages*

Each message is identified with a PauseID (Section 5.2 of [36]) following the design of a numbering sequence. The value of a PauseID is incremented by one in modulo arithmetic every time a new "pause and resume operation" is satisfied. A "pause and resume operation," in its turn, is recorded once a paused stream in the RTP sender is resumed. Logic in the RTP stream sender is organized using four distinguished states: Playing (or Sending), Pausing, Paused, and Local Paused, as seen from Figure 2.10.

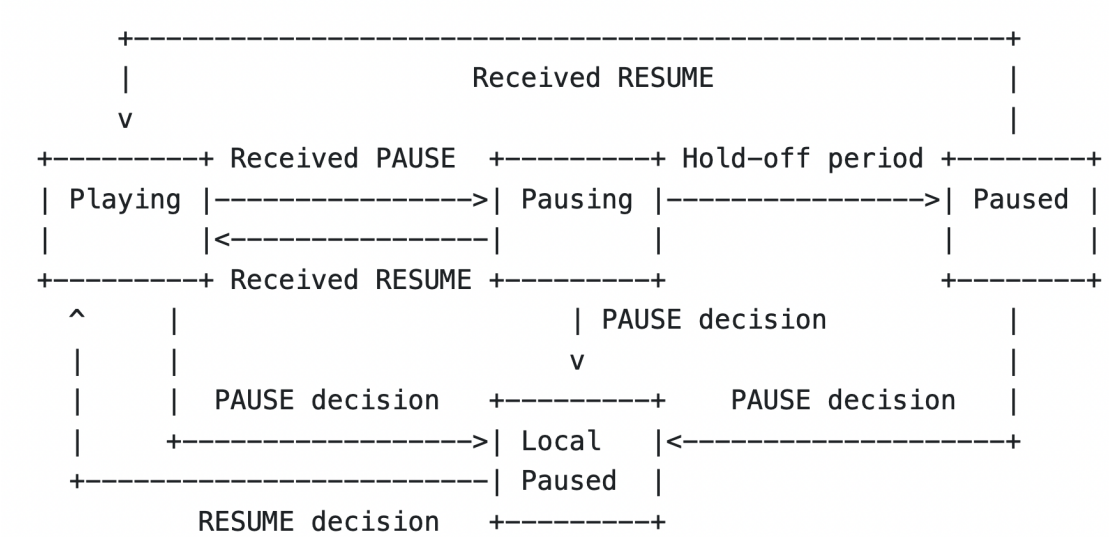

Figure 2.10. *RTP Pause States in Sender. Adapted from [36]*

The basic usage of pause-resume can be seen in Appendix A.2

2.3.3.6 Retransmission

As another choice to achieve transmission reliability, *Retransmission* (resending packets only if they are lost or damaged) offers an active and conditional approach compared to *Redundancy*. Regarding the IETF standard, the effects of retransmission are illustrated in RFC2354 [39] along with FEC and interleaving. An RTP implementation must acquire extended functionalities in both the signaling plane (to communicate control information related to retransmitted packets) and The data transferring plane (to send the lost/damaged packets again and distinguish the resent packets). In addition, the proposed solution must also conform to existing specifications. RFC4588 [40] specifies a solution using the in-band AVPF RTCP RFC4585 for control/feedback, out-of-band SDP for types association, and a separate RTP stream as the data transferring solution. The separated retransmitted stream can be allocated in the same RTP session using SSRC-multiplexing or in a different RTP session using Session-multiplexing. The former option leans towards flexibility, while the latter focuses on resource saving.

A retransmission packet (format shown in Figure 2.11) contains the following field:

RTP Header (32-bit) conforms to RFC3550 for integrated RTP functions.

OSN (16-bit) or Original Sequence Number is associated with the original packet.

Payload (n-bit) Original RTP Packet Payload

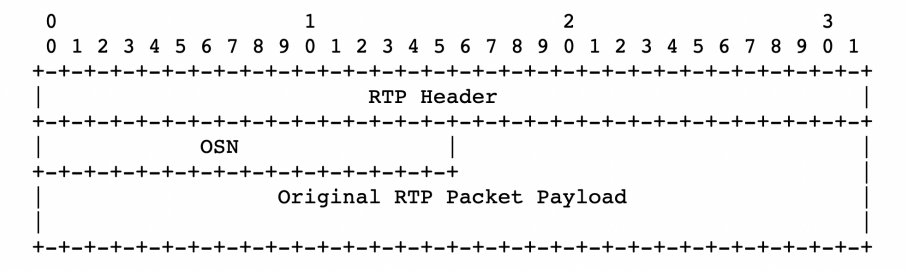

Figure 2.11. *RTP retransmission packet format. Adapted from [40]*

After retransmitting a packet, the packet association must be done so the receiver can benefit from that packet. If session multiplexing is applied, the original and retransmitted stream are associated via the exact value of SSRC in two different sessions. If the implementation is SSRC-multiplexing, the CNAME field in two RTP Headers is the initial attempt for packet association. If the CNAME cannot associate the two streams, additional information, such as retransmission packet reception, must be used. However, there are cases like multicast where even the additional information can not provide a confident result; a higher architectural solution must be considered. On the resilience vs delay tradeoff, a give-up time is set for the total retransmission duration to at least assure the upper bound for the delay.

2.3.3.7 Synchronization

In general, the purpose of synchronization is to maintain the temporal relationship of the presented data. A popular example of synchronization in AV streaming applications is lip-syncing, in which the audio and video content is aligned. Synchronizing in this example is necessary to re-construct the content that matches the user's expectation of the auditory and visual information matching. There are two main approaches to achieving the target of synchronization:

- Network level: Network synchronization synchronizes the clocks of all stream-generating devices in the network. This is usually done via protocols like Network Time Protocol(NTP) defined in RFC5905 [41] or Precision Time Protocol(PTP) defined in RFC8173 [42].
- Streams level: Stream synchronization synchronizes the received streams from the devices. This approach is operated on the application level using temporal information from the first approach.

Despite both approaches playing an important role in data consistency and integrity, the second approach is focused on this thesis work. In multiple comparative studies([43], [44], [45]), groups of authors provided some handy terminologies for multimedia streams synchronization which could be extrapolated at a certain level for stream synchronization of other data kinds. The modified version of these terms are listed below:

- Synchronization (or sync in short) is a process to maintain the temporal coordination and organization of one or multiple streams.
- Event-based sync is a type of sync technique that focuses on the orchestration of multiple streams. It ensures a triggered event results in packet transmission of the right streams.
- Stream-based sync is a type of sync technique that aims for the correct temporal order of the "playout" stream(s). This sync technique is meant to align the received streams based on their packet's temporal information.
- The subsets of stream-based sync can be listed as:
  - Intra-stream sync
  - Inter-stream sync
  - Inter-destination sync or Group sync
  - Interactive sync

  The first subset handles the temporal coordination for packets within a single stream. The second one copes with the temporal relationship between different streams contributing to a common task. Inter-destination sync is specialized for distributed networks as its goal is to ensure scattered receivers can play out streams regarding the same temporal coordination. The interactive sync goal is to guarantee multiple streams from distributed sources can maintain a sufficient temporal relationship under any random and unpredictable user interaction.

As the transmitted data in an RTP/RTCP implementation is closely tied to the time-serial characteristic, the protocol supports the synchronizing function by default and in several extensions. By using a consistent rule for timing specified in RFC3550 for the `timestamp`-RTP header field, an RTP packet generator is expected to preserve the intra-stream sync. In addition, the `sequence number` and `payload type` also indirectly support the sync function because the former provides packet loss detection, and the latter notifies the type of transmitted payload containing the information about the sampling rate. However, the initial timestamp given at the start of each stream is generated randomly, so using timestamp values solely does not solve the inter-stream sync problem. A reference-clock value, delivered via RTCP SR [24], is assigned to each RTP stream. This pair of local timestamp and reference-clock values allows a participant to perform the inter-stream sync.

In this thesis, the focus synchronizing types are inter and intra-sync, so the operational details about the inter-destination sync have been moved to Appendix A.3 to avoid a lengthy explanation.

2.3.4 Drawbacks

While the design of RTP/RTCP is well-suited for real-time applications, there exist certain drawbacks and room for improvement:

Lack of QoS and Timely Delivery

RTP, by default, does not provide any mechanism for QoS or Timely Delivery, but usually relies on RTCP for these functionalities. Alternatively, other solutions in other stacks can also be used if they deliver the same control/feedback information between participants. A standalone RTP deploying system could be insufficient for QoS-demanding and Timely-delivery-demanding applications.

Complexity on Endpoint

As a tradeoff for function modularization, participants are expected to coordinate data-transferring tasks and their reports during the operation. In any RTP architecture, the complexity significantly increases at the components where the signaling and data-transferring function intersects. In addition, multiplexing is required for applications using RTCP in the control plane to reduce port usage and associate the transmission report with data packets.

Network-aware Application

Thanks to the blurred separation between the transport and application layers, an RTP application is granted access to transport-layer resources and multilayer network monitoring means. On the other hand, those extended capabilities put more weight on the development of a real-time application in comparison with applications using different solutions.

2.3.5 Reliability

The reliability of packet transmission is a common challenge for transport protocols. As the RTP stack is built on top of IP/UDP, the packets can be lost during the transmission without acknowledgment. Therefore, the IETF community proposes numerous solutions using RTP/RTCP to mitigate this problem. In addition, other candidates outside the scope of the transport layer can be applied to improve RTP reliability. Due to its importance, even if there could be some overlapping with the QoS and Timely Delivery in Section 2.3.4, this section will be dedicated to discussing the topic of reliability on RTP.

A reliable transmission generally means all transmitted packets from the sender are successfully received at the receiver side. On the RTP level, there available two solutions:

Redundancy (Section 2.3.3.3): Sending more packets by default.

Retransmission (Section 2.3.3.6): Re-sending only the lost packets.

Given the specific conditions of an RTP deploying system, a system designer can choose a suitable solution or a set of solutions to enhance reliability. If redundancy is selected, the extra-bandwidth consumption and additional processing power for stream merging have been approved. If retransmission is applied, there is less bandwidth consumption for data because a sender will only send the known lost packets. On the other hand, both the sender and receiver are assigned additional tasks to send the correct missing packets. Different aspects to be considered before applying redundancy or retransmission can be seen in Table 2.3.

Aspect	Redundancy	Retransmission
Bandwidth	Deterministic and dependent on the redundant level	Nondeterministic but can be limited by the give-up time
Latency	Depends on the stream-merging operation	Depends on a long process of packet loss detecting, resend requesting, and packet lost resending operation
Complexity	Lower and mainly within the stream merging operation	Higher and requires many additional functions
Multicast scalability	Simple in terms of implementation, but results in large bandwidth consumption	The complexity also proliferates, but it is bandwidth-wise more acceptable
Memory consumption	Lower as there is no additional buffer	Higher because the sender needs to buffer packets to resend and the sender needs to buffer packets to detect loss

Table 2.3. *Redundancy vs. Retransmission for Reliability*

Alternatively, if the lower layers allow each packet to be sent with a higher guarantee, one can also see the reliability of the RTP deploying system. Concrete examples of this approach would be technologies in the Ultra-Reliable and Low Latency Communication category of the 5G. To meet the requirement from E2E latency of 1ms with $1 - 10^{-5}$ reliability for machine control tasks to $1 - 10^{-9}$ for surgery tasks, some possible solutions

are highlighted in the work of Zexian Li on 5G URLLC Design Challenges and System Concepts[46]:

Micro-diversity: more antennas on the transmitter and receiver sides.

Using a higher reliable control channel.

Using a dedicated channel for high.

Improving the Signal-Noise Ratio by reducing the received interference from the neighbor gNBs and UEs.

2.4 SIP

2.4.1 Overview

SIP plays an important role in VoLTE networks as it is the main protocol for signaling functions required before any RTP session. Session Initiation Protocol (SIP) is an application layer control protocol to support session-related functions (including establishment, modification, and termination) and user mobility(via SIP URL and service registration). The current SIP is SIP version 2, specified in RFC3261 [47], and has been continuously updated with new functions. A wide range of applications in IP-based communications is implemented using SIP as the main control protocol due for the following reasons:

- A large community of IETF contributors supports SIP in developing a unified communication system.
- The modular development is guaranteed. SIP only provides the means to deliver the control signal for a session and does not interfere with the session description function or content delivery. A system designer can choose from other methods to describe what SIP delivers. Session Description Protocol is (SDP) often chosen for this purpose.
- SIP is an End-to-End protocol.
- Interoperability is embedded in the protocol as two or more devices can negotiate their session setting before initiating one.
- Scalability is activated by shifting the complexity from the network to end devices. Once a session is up, the impact of a SIP server is small, which expands the network's capacity.
- SIP is accessible and straightforward because it is based on many popular IP network concepts, like HTTP resembling syntax or URIs as SIP addresses.

2.4.2 Design

SIP is based on HTTP in its design. Therefore, it also involves the request/response mechanism realization of entities like clients and servers. A SIP entity that can generate SIP requests is a SIP client. Those requests can be received by another entity called the SIP server, which can answer requests by returning responses. In SIP terms, a client application and a server application are called User Agent Client (UAC) and User Agent Server (UAS). A UAC issues requests, while a UAS issues the corresponding responses. SIP Proxy or SIP Proxy Server or SIP Server is a logical entity in a SIP deploying system that handles multiple functions:

- Routing requests to users.
- Implementing provider call-routine policies.
- Providing user authentication and authorization for services.
- Providing additional features like voice mail.

A SIP Proxy can be in two modes: "stateless" or "stateful". A stateless SIP Proxy acts as a forwarding element and discards the information about a message as soon as that message is delivered. If a SIP Proxy is stateful, all transactions handled by that Proxy are remembered for future processing. A stateful SIP Proxy can also discard unnecessary transactions. Therefore, a SIP Proxy can switch between its two operation modes. To clarify two Server definitions in SIP, a SIP Server is defined in the network scope as an entity, while the User Agent Server refers to the role of a Server in an application using SIP. The last two essential elements for a communication system using SIP are the SIP Registrar and SIP Redirect Server. A SIP Registrar maintains the information associated with a user and passes this information to other entities in the network. A SIP Redirect Server provides additional information (like the alternative URI or phone number) for the originator of a request in case a user can not be reached using the previous URI.

SIP Message

The SIP message is the general term for SIP request and SIP response. Each SIP message contains the following:

- A type indicating line:
  - Request line for SIP request in the format: `SIP-method SIP-URI SIP-version`. Example: `INVITE sip:trung.van@tuni.com SIP/2.0`.
  - Status line for SIP response in the format: `SIP-version Status-code Reason-phrase` Example: `SIP/2.0 180 Ringing`.
- A SIP header may have one of the popular fields listed in Table 2.4. A header contains compact but important information to quickly describe a SIP message without

processing the content in the SIP body. SIP headers format is `Header-field: Header-value`. For example: `To: Trung <sip:trung.van@tuni.fi>`

- A SIP body (optional) contains an object describing operations on a session.

A series of correlated SIP messages form a SIP transaction. As expected, SIP is a transactional protocol, meaning SIP entities interact with each other using these SIP message exchanges.

Field	Common form	Function
`Call-ID`	A generated string	Uniquely identify a series of SIP messages
`Contact`	A URI	Directly locate a user
`Cseq`	An integer followed by a method name	Order repeated requests in the same session and match requests to responses
`From`	A display name followed by a URI	Identify the initiator of the request, could be the address-of-record
`To`	A display name followed by a SIP URI	Identify the recipient of the request
`Route`	A list of URIs separated by commas	Indicate the forced proxy routing applied on a request
`Via`	A list of proxy URIs involved to handle a request	Store the path of a request to avoid looped routing

Table 2.4. *Popular SIP headers*

SIP Request RFC3261 defined six main types of SIP requests serving different purposes. In general, all protocol functions in SIP are initiated using SIP requests. Until now, several new types have been added to the initial set. The popular SIP request types can be seen from Table 2.5.

Request type	RFC ref	Function
REGISTER	RFC3261	Register the URI listed in `To`-header field with a location server
INVITE	RFC3261	Invite users to join a session
ACK	RFC3261	Acknowledge the reception from the final response to an INVITE request
CANCEL	RFC3261	Cancel a pending transaction
BYE	RFC3261	Indicate the leaving of a session participant and terminate a session if there is only one participant
OPTIONS	RFC3261	Query information about a server's capability
UPDATE	RFC3311	Modify session's parameters without affecting the state of a dialog
REFER	RFC3515	Instruct the recipient of this REFER to issue a new SIP request
SUBSCRIBE	RFC6665	Create a subscription to receive notification about an event from a notifier
NOTIFY	RFC6665	Notify a subscriber when a new event occurs
PUBLISH	RFC3903	Publish an event to a notification server
INFO	RFC6086	Deliver information about a session without modifying its state

Table 2.5. *Popular SIP requests*

SIP Response

A SIP response is sent by UAS to UAC to indicate the result corresponding to a SIP request. Different classes of responses are numerically coded similarly to HTTP codes. A summary of them can be referred from Table 2.6

Range	Class	Function
1xx	Informational	Indicate a request is valid and being processed
2xx	Success	Indicate a request has been successfully completed
3xx	Redirection	Indicate a redirection is required for a dialog
4xx	Client Error	Indicate a failure of a request from client side
5xx	Server Error	Indicate a failure of a valid and received request from server side
6xx	Global failure	Indicate a request can not be responded by any server

Table 2.6. *SIP response classes*

2.5 VoLTE

2.5.1 Overview

VoLTE is the realization of the RTP plus SIP stack to provide voice services popular in cell phones today. Therefore, it is necessary to provide a brief overview of VoLTE in this thesis. VoLTE is an abbreviation for Voice over Long-Term-Evolution whose name conveys quite well its primary function of delivering VoIP calls over LTE. Subsequently, LTE is defined as a high-speed wireless communication standard (3GPP Release 8 and 9 document series) for cell phones and data terminals.

VoLTE was first introduced as an industrial initiative called One Voice in 2009. This initiative aims to provide a minimal profile for manufacturers and service providers to enhance interoperability in the race for voice service over the IP Multimedia Subsystem (IMS). IMS is a standardized architectural framework for delivering multimedia services over an IP network. In 2010, GSM Association officially adopted One Voice and renamed it VoLTE. As there are multiple associations involved in the standardization, a proper VoLTE implementation must satisfy both the specifications from the GSM Association for VoLTE and 3GPP for LTE. Some critical differences related to VoLTE between these two standardizing entities can be seen in Appendix A.4

VoLTE is not limited to LTE, as WiFi and 5G have been included in the connectivity. Based on the Ericsson Mobility Report of 220 [48], VoLTE has been activated for over 280 net-

works. The current number of VoLTE subscriptions is about 4.8 billion, and Figure 2.12 suggests this number can reach 7 billion in 2027.

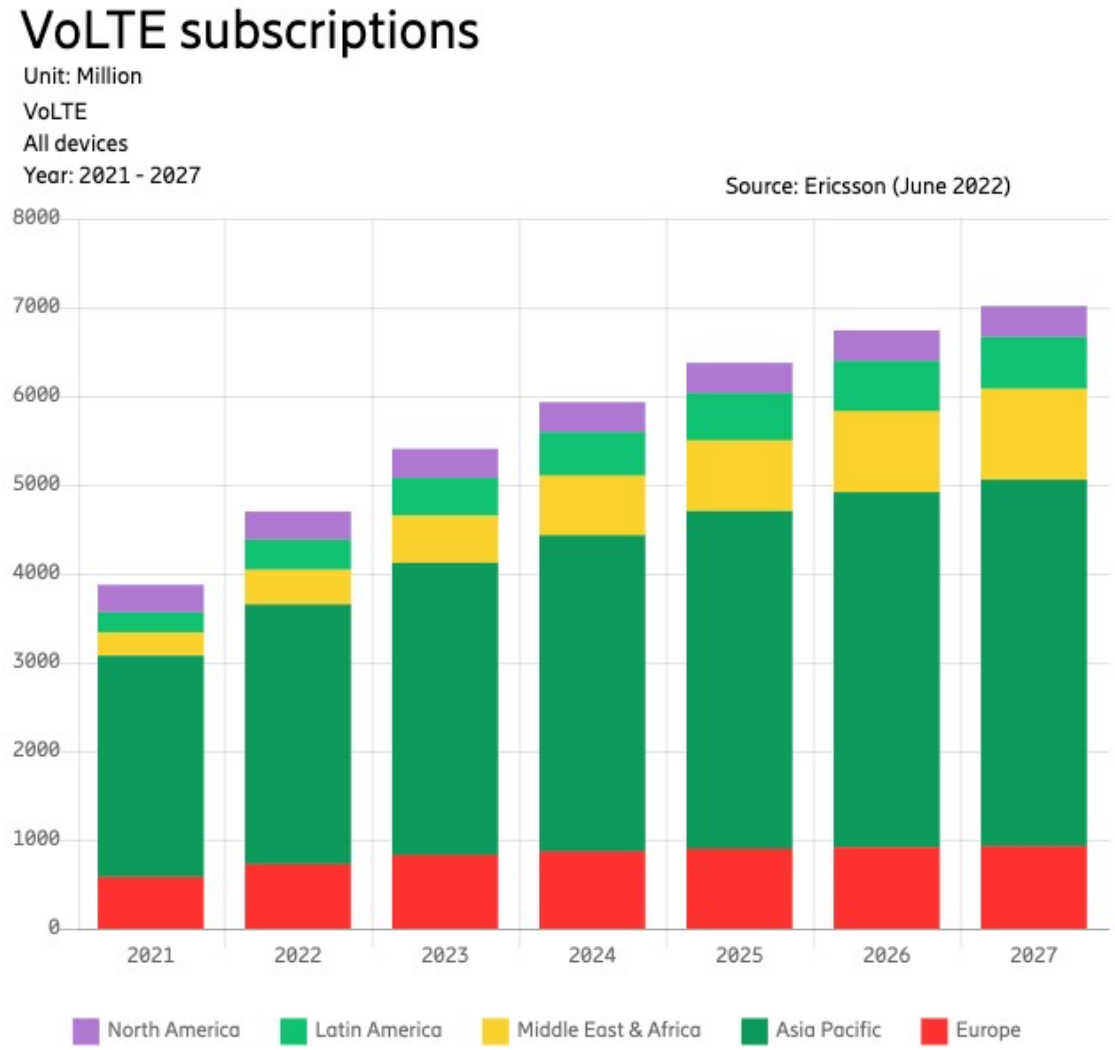

Figure 2.12. *VoLTE Subscription from 2021 to 2027. Adopted from [48]*

From Ericsson Report on Mobile Subscriptions, the number of 5G and LTE subscriptions is projected to reach 9 billion in 2027, as seen from Figure 2.13. This includes VoLTE services in about 90 percent of the future 5G/LTE subscriptions. This could be attributed to factors like the wide range of connectivity supported by VoLTE, the Circuit-Switch network obsolescence, or the improvement in operators' services to compete with other VoIP OTT(Voice over IP Over-the-top) applications.

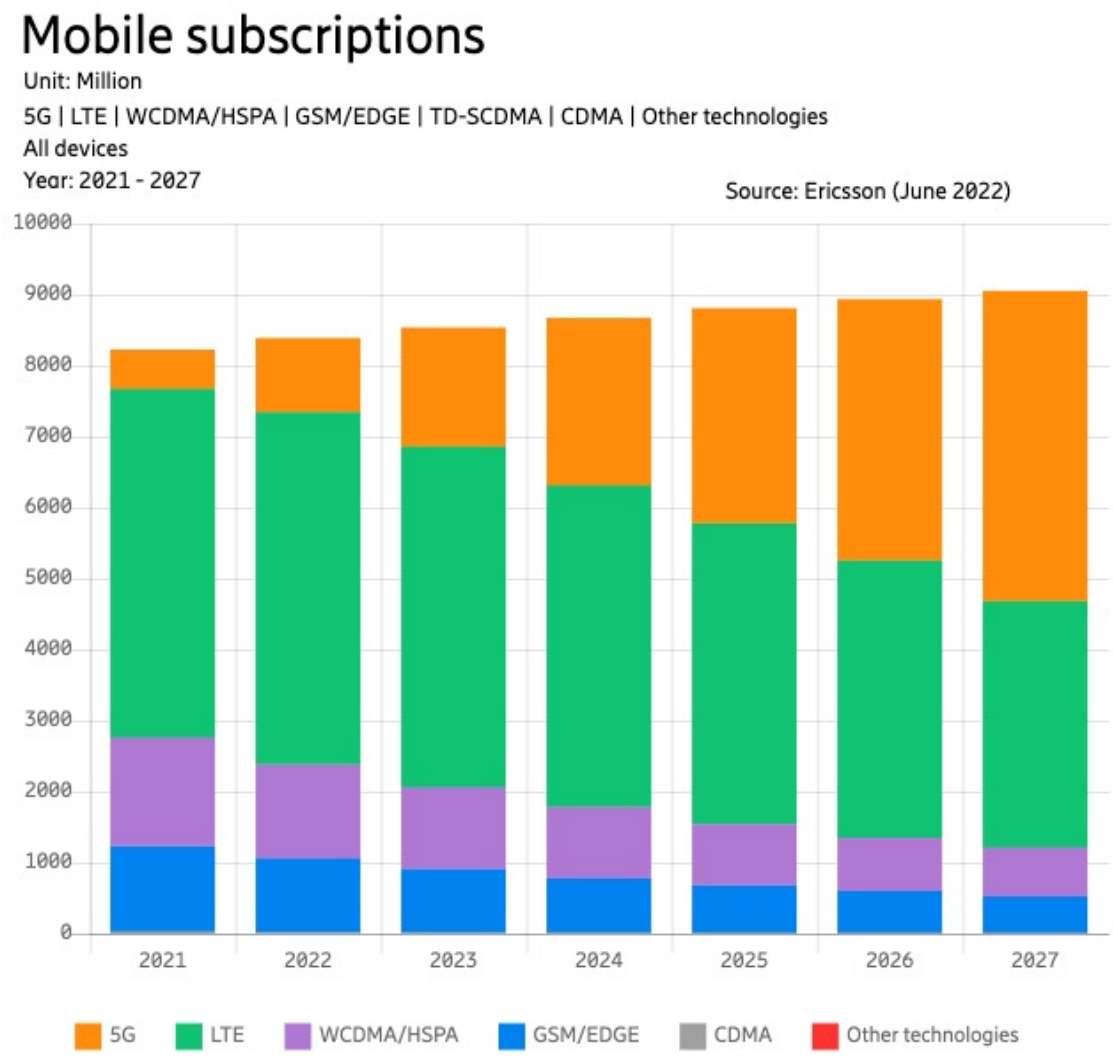

Figure 2.13. *5G and LTE Subscriptions from 2021 to 2027. Adopted from [48]*

By using the VoLTE services, a user can take advantage of different VoLTE properties:

- A reliable and well-maintained connection.
- A constantly improved connection in terms of coverage and connection quality.
- A user-friendly experience and no additional application required.
- A short service connection setup time.
- A lower energy consumption level compared with other VoIP solutions.
- An inter-operator service by default.
- Supported various device types.
- Emergency support and fallback capability.
- A fixed and predictable cost.

From a provider's perspective, continuing the support for current VoLTE services and expanding them with new research are both critical. The two most obvious reasons are

re-purposing the investment in network infra and securing a stable income from subscriptions. In addition, an operator can also up-sell their customers with:

- Hardware and connectivity in one purchase: An operator can offer an individual an offer that includes both hardware(smartphone, smartwatch ...) and their VoLTE package.
- Extending the current usage: An operator can offer more premium features in their services to the current users to compete with others.

2.5.2 Functionalities and Architecture

For data transmission between two devices, two minimal requirements must be met: The connectivity between them and the actual data transmission on that connectivity. The network core domain handles both requirements during the circuit switch network. In VoLTE, access mobility and connectivity are handled by the Evolved Packet Core (EPC) in the Evolved Packet System (EPS). At the same time, the IMS is responsible for the calls and media control. This functional division results in the architecture of VoLTE, whose simplified version can be seen in Figure 2.14.

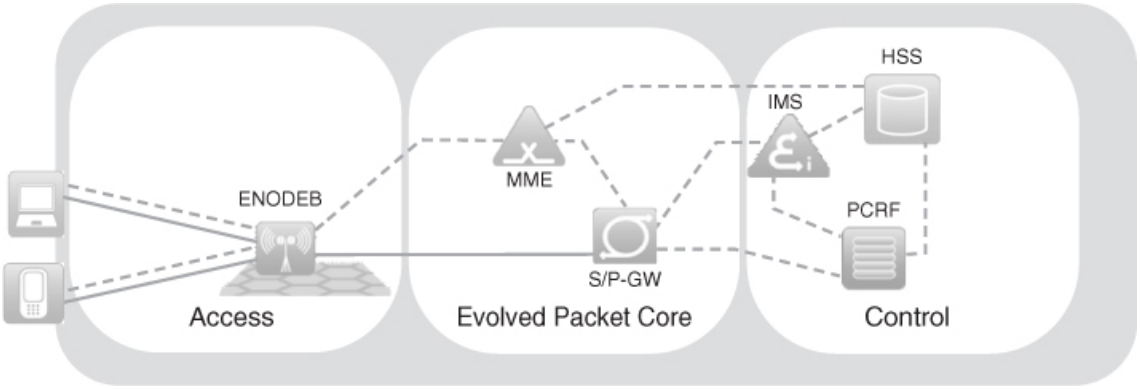

Figure 2.14. *Simplified VoLTE architecture. Adopted from [49]*

A VoLTE architecture spreads into three domains: Access, EPC, and IMS. As a consequence, a collection of VoLTE main functionalities can be clustered into three main groups based on GSMA's official "IMS Profile for Voice and SMS" [50]:

- Provided at the Access domain:
  - Packet scheduling and QoS differentiation to prioritize packet transmission.
  - Mobility enables seamless handover and fallback in either intra or inter-system scenarios.
  - Power saving feature on User End-devices (UE) using discontinuous transmission and discontinuous reception (DTX/DRX).

  - Positioning using multiple solutions.
- Provided by the EPC network:
  - Identification for LTE Subscriber
  - Establishing the Packet Data Network (PDN) connectivity (always-on IP connectivity between a user and a PDN) for the VoLTE User.
- Provided by the IMS network:
  - Multiple Identification functions for users(in both public and private domain), subscriptions, devices, public services, and network entities.
  - Service Provisioning by a *SIP-based service invoking architecture* on the Packet Switch network.
  - Rich communication with supplementary services on top of the basic media capabilities.

A more detailed VoLTE architecture (as can be seen in Appendix A.4) can be found in the GSMA official document for VoLTE service description and deploying guideline [51].

2.6 Conference

2.6.1 Overview

A conference (or teleconference) refers to a multiparty communication session that involves sharing voice, video, instant messages, or data of other types. This application is popular in both the enterprise sector and the sector of individual Internet users. Due to the demand for this application, it has been standardized in 3GPP TS 24.147 [52] to provide the details for implementation on the IMS Core Network. This document glues different technology fragments discussed above (SIP, RTP, RTCP, VoLTE) into a complete conference application blueprint, and commercial implementations have been built based on that. The conference's infrastructure reusing and conferencing characteristics motivate this section to describe its rough architecture. The content of this section can be used to understand conference system design without reading all of its specifications. Like many other applications implemented on IMS infra, conference functions are separated into two planes: Signaling and Data Transferring. Conference system design mainly focuses on the Signaling plane to manage the multi-participant sessions. Challenges in Data Transferring are already common in media streaming and can be overcome by applying the available solutions. The design of a conference system is discussed in the following subsection.

2.6.2 Design

The 3GPP TS 24.147 instructs the use of the SIPPING framework, which in its turn is specified using multiple IETF documents, including:

RFC4353 A Framework for Conferencing with the Session Initiation Protocol (SIP)

RFC4575 A Session Initiation Protocol (SIP) Event Package for Conference State

RFC4579 Session Initiation Protocol Call Control - Conferencing for User Agents

RFC6665 SIP-Specific Event Notification

Generally, a conference based on the SIPPING framework follows the tightly-coupled model. An illustration of a tightly-coupled conference can be seen in Figure 2.15. Each conference participant maintains a signaling relationship with a conference server. Streaming data from each participant is also sent to a mixer co-located at that server.

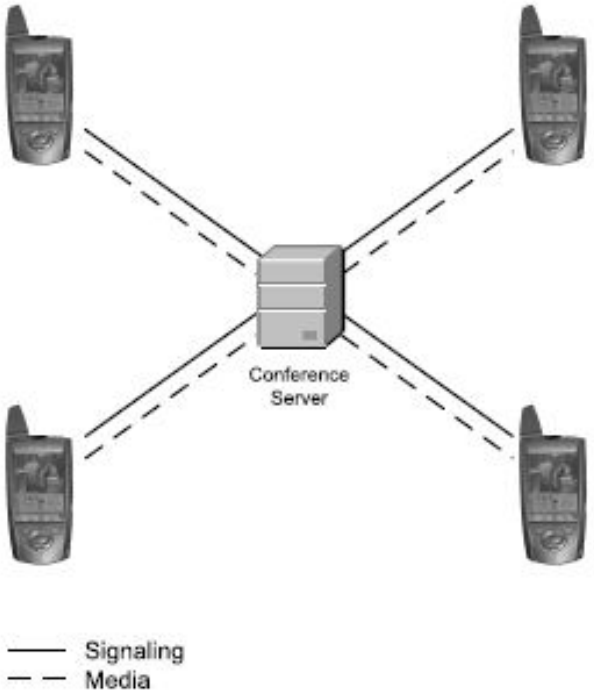

Figure 2.15. *Tightly-coupled conference model. Adopted from [53]*

2.6.2.1 Components

RFC4353 introduces several fundamental components of a conference service used in both IETF and 3GPP documents:

Focus is the central component responsible for orchestrating the whole conference system to make sure participants can receive media streams. In SIP terms, Focus is a user agent and is addressable by a conference URI. All conference participants interact with the Focus by SIP dialogs.

Conference Policy Server is a logical function responsible to manage the conference policy.

Conference Notification Service is a logical function acting like a notifier RFC6665[54] to notify its subscribers about the changes in the conference state.

Mixer is similar to its definition in RFC3550 but allows other data types beside A/V.

Conference Server is a physical server based on Focus (at the minimum), Conference Policy Server, Conference Notification Service, and Mixers.

Participants must be capable of performing at least SIP user agent functions.

The communication between these components can be seen in Figure 2.16.

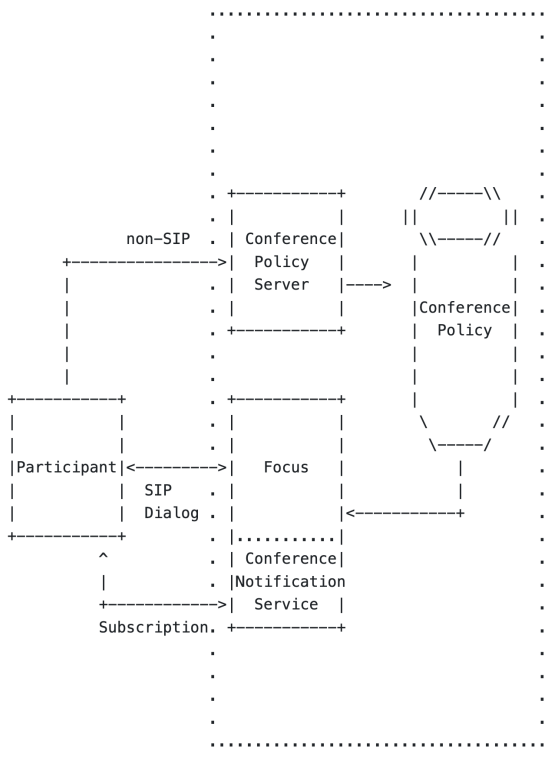

Figure 2.16. *Model of Functions involved in a Conference using SIPPING framework. Adopted from [55]*

Conference-related information(Conference State) is maintained in the form of an XML object-Conference Object(an overview in Figure 2.17) in the Focus. The RFC4575 specifies the content of a Conference State, a SIP conference event package for that conference state, and a mechanism to subscribe to the conference state. This *Subscribing to/Receiving notification of* mechanism is responsible for the bidirectional arrow between the Conference Notification Service to Participants in Figure 2.16. The RFC4575 also provides a partial notification mechanism to avoid re-sending the unchanged or unnecessary content in the conference state. Another observation that can be made from Figure 2.16 is that Mixers are not included in this illustration. To explain, a Mixer is not involved in

the signaling process and only receives media streams from Participants and instructions from the Focus to generate output streams. In addition, a Mixer can be placed within the Focus, or many Mixers can be distributed near Participants depending on the chosen architecture.

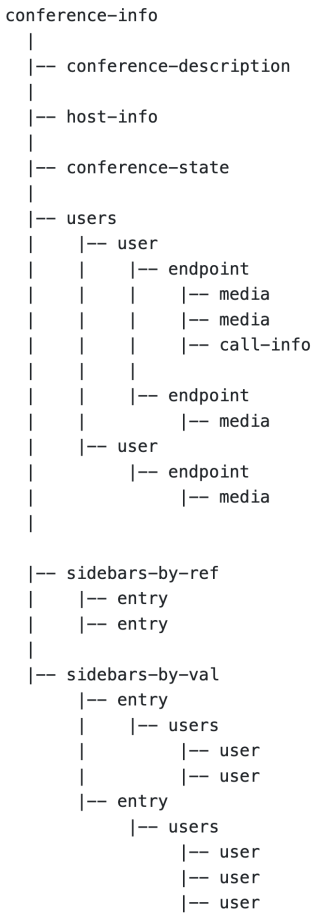

Figure 2.17. *An overview of a Conference Object. Adopted from [56]*

2.6.2.2 Operations

RFC4579 describes the operations of the Focus and Participant UAs applied for a tightly coupled conference. For a more abstract view, some common operations with a short description can be listed:

Creating Conferences

A conference creation simultaneously triggers the construction of multiple entities, including: a Focus to maintain a SIP relationship with participants, a unique URI to identify that Focus, and a Conference Policy with default rules.

Adding Participants

A participant has two ways to join a conference: either by adding himself or having another

add him in. The first option involves an INVITE request to be sent to the conference URI from a participant. The conference server, after that, decides to allow the user to join based on its conference policy. The latter option requires a SIP REFER request from a participant. This request can be sent to another participant to introduce him to Focus and vice versa.

Removing Participants

A similar approach to **Adding Participants** can be applied for this operation by replacing the INVITE request with a BYE request. A participant can be removed from a conference by himself, by the Focus, or by another participant. In any case, the Focus must remove the media stream from/to that participant.

Destroying Conferences

When a conference is destroyed, the entities created to support its work are destroyed simultaneously, namely the conference URI and the conference policy. The existing subscriptions must be terminated, and BYE SIP messages must be sent to the active Participants in a conference before the destruction process.

Obtaining Participants Info

The information about other participants within the same conference can be obtained by subscribe/notify or fetch mechanism.

Adding/Removing Media

A conference may change its media set, and participants can select what streams to receive or reject. This operation is done via SIP re-INVITE request.

Announcing in Conferences

In principle, the announcement application is an automated participant in the conference. Its sent data only serves the informative and interactive purposes of the conference. This information can be blended into the shared communication with all participants or tailored for an individual participant.

2.6.2.3 Architecture

Different realizations can be formed by varying the amount and arrangement of conference components. In this subsection, the Centralized Server architecture is shortly described with its illustration adopted from RFC4353. Other architecture descriptions can be found in the Appendix A.5

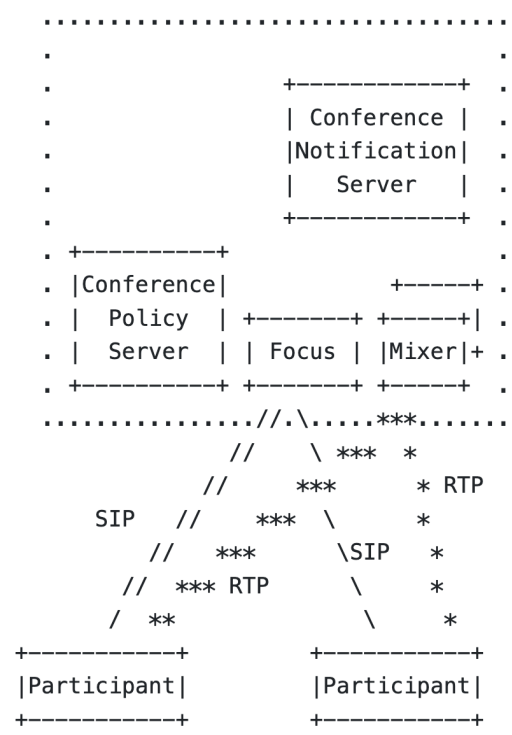

Figure 2.18. *Centralized Server Model. Adopted from [55]*

In this realization, a single physical server handles signaling and data transferring. This Centralized Server is responsible for many functions: a Focus, a Conference Policy Server, and Mixers.

2.7 Pub Sub

2.7.1 Overview

Today, one can easily find the term PubSub when working with data distributing systems. PubSub is a software architectural design pattern based on the asynchronous messaging model. A PubSub system generally contains publishers who send messages and subscribers who receive messages. Instead of sending messages directly to a subscriber, a publisher sends its messages to a middleware that will distribute them to interested subscribers. Related messages, e.g., from the same publisher, are organized under the name of a topic. In a PubSub implementing system, a publisher and a subscriber of the same topic are not required to synchronize their operations. A broker handles the routing and coordinating tasks between publishers and subscribers. More technical details are described in the following subsections. *Broker pattern*-the most popular pattern is chosen to demonstrate in this section, while various patterns exist to implement a PubSub system. Other technical details can be found in the following subsections.

2.7.2 Primitives

There are several primitives prevalently used in the paradigm of PubSub, and some of them can be seen in Figure 2.19:

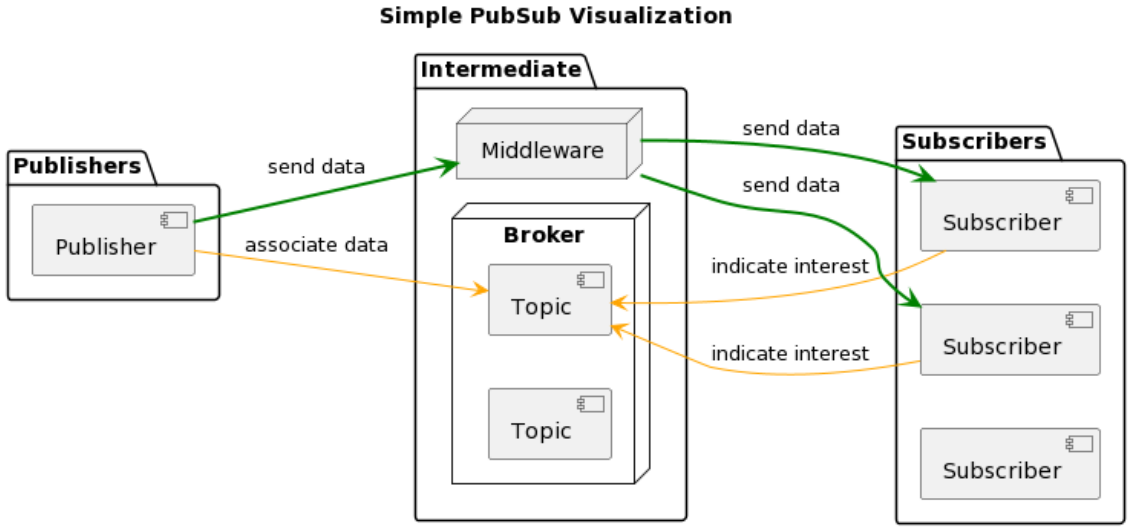

Figure 2.19. *PubSub primitives example*

Event is a discrete state transition that happened in the system and can be either atomic or composite(as a combination of atomic events). An *Event message* is a container for Event traffic within a system. When one mentions data in a PubSub system, he usually means Event messages.

Notification is a response of the system towards a detected event for the receiving agent.

Topic is a named resource from which correlating messages can be sent and retrieved. It addresses the data under its name.

Subscriber is an agent interested in a specific topic. A *Subscriber*, after successfully making a subscription to a topic(could be further specified using additional conditions), will receive the valid Events if they arrive.

Publisher is an agent capable of detecting or producing events and associating its data with a topic.

Broker is an intermediate agent with various functions like data routing, notification routing, or topic management. A broker can be capable of data forwarding or yield that function for other *Middleware*s.

2.7.3 Principles

As a paradigm, there are several central design principles required to create more sophisticated rules and practices:

- The interface of a PubSub-system is logically centralized, and this interface abstracts away the internal organization of the broker(s).
- The communicating agents are decoupled from each other.
- A filtering mechanism (*upstream filtering*) and distributing mechanism(*downstream replication*) are required. Both processes should be as close to the southbound as possible to reduce unnecessary traffic.
- The system has to be modular and extensible to stay flexible.

2.7.4 Properties

Some popular properties of a PubSub system are:

Spatial decoupling: Decoupled publishers and subscribers handle their complementary functions using separated resources on two ends of the system. Therefore, the publishers and subscribers are unaware of each other, and the number of subscribers for a topic does not affect the publishing behavior.

Temporal decoupling: All involved agents are not required to be active simultaneously for PubSub-operations. The sending agent does not have to be active as long as the event message/notification has been successfully delivered to other intermediate agents.

Synchronization decoupling: Subscriber and publisher can operate other tasks/processes while concurrently performing their PubSub-functions. No synchronizing tasks are required for the publication and subscription to be sent.

3. IOT-STREAMING-SUITED REALIZATION

This chapter focuses on the practical application of RTP and its extensions for IoT streaming use cases. The chapter explores how RTP can be effectively utilized to ensure the reliable and efficient streaming of IoT data, considering factors such as data size, network bandwidth, and latency. The chapter also investigates how SIP can be leveraged to facilitate IoT streaming and identifies relevant findings that require additional explanation on SIP. Additionally, the chapter examines the relationship between RTP and VoLTE and identifies potential opportunities for using VoLTE capabilities in IoT streaming applications. To make the practical application of RTP and related protocols more understandable, the chapter will include use-case examples demonstrating how these technologies can be applied in real-world scenarios. The findings from this chapter and the use-case examples will be applied in the subsequent Prototype Implementation chapter to create a practical, real-world example of how RTP and related protocols can be used effectively in IoT streaming applications. By providing practical insights and use-case examples, this chapter serves as a valuable foundation for the subsequent Prototype Implementation chapter.

3.1 Using RTP/RTCP

Given the functionalities described in Section 2.3.3, numerous use cases of IoT streaming are discovered and the ratios of RTP adoption over novelty vary case by case. This section is dedicated to illustrating the realization of IoT streaming from an RTP perspective using some concrete examples.

3.1.1 Built-in features

Using RTP for data transferring, multiple basic functions are enabled by default. The following list shows some of them that are relevant to this thesis work:

- Packet loss detection and stream reconstruction using the Sequence Number.
- Preserving the temporal information, thanks to the timestamp value, and inter-stream synchronizing based on that value.
- Identifying packets belonging to the same stream.

- Extendability via the extended header.
- Identifying multiple streams contributing to a more complex stream using the CSRC.

As some built-in functions are covered by or closely related to the other functions in the coming subsections, they are explained together.

3.1.2 Synchronization

Synchronization is a critical function for a streaming system, and its importance even escalates in some IoT contexts. Like other applications involving time-serial data, the received stream must be aligned with others to exploit their temporal relationship. With a proper synchronizing function set, an IoT streaming system is enhanced in multiple ways:

- IoT data from various sources can be combined to provide more complex application logic.
- High temporal resolution data can be applied on top of low temporal resolution data to increase confidence, if needed, before making any critical decision.
- If packets in an IoT stream are lost or damaged, other streams used along with that stream can partly cover the missing information. This could be useful for unreliable and constrained networks whose devices are not capable of retransmission.

The heterogeneity and network scale in IoT applications like WSN result in a large number of streams to be synchronized and the sync-setting that could be required within a network. These two challenges may require a different distribution of synchronizing points to fit the devices and network constraints. Besides, the stream processing speed in electronic devices increases the upper bound for the number of events in a stream. For example, some sensors operate at frequencies of thousands of Hz, and different manufacturers may use their proprietary settings. Therefore, a system designer must also consider specific devices' processing frequency to select intra-stream and inter-stream sync strategies. In addition, different qualitative measurement methods for IoT streaming services are beneficial to improve service quality. Despite several QoE assessment methods have been proposed as in [57], [58] or [59], QoE for IoT streaming is still not yet standardized.

Example - Analytical Streaming

In the realm of IoT, there are scenarios where devices' high temporal resolution update is yet available, or the need for that sensory data is occasional. Some examples can be envisioned as follows:

- A healthcare service would like to offer remote diagnostics to support emergency cases or more convenient operations for simple tests. Vital bio-signals are streamed but only for a short period. Of course, these signals must be synchronized to rep-

resent the patient's condition as much as possible. The necessary device to extract those signals can be equipped for an emergency medical service in an emergency case.

- In less severe cases, like for a simple test or checkup, a sensory device can be a module attached to a UAV for home delivery. This convenience could encourage the users to have their checkups more frequently with high temporal resolution. If there is a small percentage of a decrease, the patient could permit a higher volume of data to be collected before going to the next check. Thanks to the additional data and its early availability, a doctor could improve the final diagnostic.
- A conservation organization would like to collect data on animals or plants in a large area. The time-series data is beneficial, but maintaining the whole network infrastructure for that purpose could be costly. A possible solution could be a short-range but high-temporal resolution device to collect streaming data from multiple targets in its coverage. This application needs a module to stream the field-collected data to the nearest server. The courier could be UAVs or that organization's employees on duty trips. A relevant prototype was made in [60] to prove the feasibility and efficiency of autonomous drone-based data harvesting.

The streaming module in the mentioned examples is realized as an RTP/RTCP module. This device must prioritize transmitting the data to the server rather than performing on-device synchronization. The synchronizing point is located at the server after the data is duplicated and distributed to multiple interested services. Only intra-stream sync and inter-stream sync are required in these applications.

A more concrete RTP procedure for a home healthcare service can be described below. When a request for at-home service has been made, a UAV is sent to the patient's address. The user can start collecting their bio-signals by integrating multiple sensors into the UAV. The SIP module starts to negotiate an RTP session for the streaming. A set of formats for bio-signals must be approved as a prerequisite for this use case in RTP. A single session can transmit multiple streams of different signal types. Each signal is identified as a source with a specific SSRC value to distinguish it from the other. The RTP module in the UAV acts as an RTP sender that keeps generating multiple RTP streams and sending them to an RTP receiver at the analysis provider/hospital. The synchronizing point is located on the server side to make sure the vital data is backed up and will be effectively synchronized with sufficient computing power. The RTCP feedback reports communicate the transmission quality. On the server side, RTP packets are received from the same address:port values are sorted by their SSRCs and CNAMEs to reconstruct the bio-signals. After that, these signals are synchronized using their timestamps before being fed into a multi-model diagnostic program. The synchronization error depends on the component streams in the sync group. For example, electro-physiological signals (EEG,

EMG, ...) must have a stricter requirement than the cardiac signal. In addition, this home service can also provide the option to forward the streams from users' healthcare wearables. These data can be sent to the RTP module using short-range connectivity like Bluetooth and then steamed to the server as an RTP stream.

3.1.3 Multiplexing

For IoT, some properties of multiplexing are remarkably valuable. Some of them can be listed as:

Reducing the number of addresses and ports.

Reducing overhead stream and session set-up time.

Using a single Network Address Translation (NAT) binding for all multiplexed streams.

Reducing the complexity in end-to-end communication management.

In addition, by introducing IoT data type to RTP streaming, two new multiplexing scenarios occur: multiplexed stream with only IoT packets (*IoT multiplexing*) and multiplexed stream with both IoT and audiovisual packets (*IoT-AV multiplexing*).

IoT multiplexing

In many systems that take advantage of analytics, it is typical to store the streaming data for analyzing purposes. It could be beneficial to store the multiplexed streaming data as it comes, and the demultiplexing is on-demand rather than by default. For applications that are analytics-focused, less time-critical, or on-demand, this adjustment can reduce the cost and more efficiently allocate the processing time for streams.

Example - Smart City

Smart City is a grand scope application of IoT where multiple dense networks continuously generate massive data. Multiplexing IoT streams is vital for these applications to gain practical bandwidth usage and session management. A stream can be associated with each other via different multiplex methods (Table 2.7):

- Their common physical original device (Multiple Data Types, One Session, or One Endpoint as One SSRC),
- Their common geographic location (One Data Type Multiple Sessions),
- Their common type of data being transmitted (One Data Type Multiple Sessions).

The multiplexing streams association is optimized using additional information on how this data will be used. Using the session-multiplexing with 32-bit SSRCs, a sufficient number of participants can operate and identify themselves in the network.

Example - Multi-signals Processing

Multi-signals Processing application is another field of IoT streaming that takes advantage of the multiplex. As Machine Learning techniques are gaining popularity in signal processing, the need for correlated data as inputs for a model is becoming more significant. After being trained with a certain number of inputs, a model is better to operate with the same input in the interference phase. Suppose the input comes from a group of RTP participants. In that case, their streaming data should be multiplexed not only for the benefits mentioned in 2.3.3.2 but also to reduce the number of processing points in the system. For continuous training models where the parameters are updated on the flight (during operation time), a practical issue is to handle the missing elements in a set of inputs. Multiplexing incoming streams is an approach to force the shared-fate property on the model's input if one wants to avoid using past or extrapolated values. The transient loss is expected to be eased by a noticeable amount of transmissions that can be done in the same period.

EEG signals processing is a representative example of this kind of application because the brain signals are probed from multiple points on the user's scalp but are usually used in the same model. An EEG device requires an intermediary like a mobile phone or a laptop. However, streaming the EEG signals as RTP multiplexed packets directly to a powerful processing unit nearby is a promising solution. The benefits could be increased mobility, computing resources access/upgrade, and unlimited recording as a service.

IoT-AV multiplexing

Some applications process AV and IoT data simultaneously, like XR, drone control, or enhanced monitoring systems, where the multiplexing of all streams is convenient. For service providers, IoT-AV multiplexing adds value to their current AV product, quickly rollouts new IoT products, and reuses the invested AV streaming infrastructure.

To achieve a functioning implementation of IoT-AV stream multiplexing using RTP, the desired conditions are:

- Devices operating multiplexing/demultiplexing must be able to handle the involved data type.
- The mechanism to arrange the multiplexed packets must be agreed upon and feasible for multiplexing/demultiplexing devices.
- If packet-loss recovery is wanted: at least one RTP mechanism should be applicable if no new mechanism is provided.
- The multiplexed packets must remain operable for other RTP functions required for this application.

A discussion on the first two bullet points may start with an essential factor affecting the audiovisual data format: human sensory ability. There are certain upper limits for the human sensory system, like 60-240 frame-per-second for vision or 20-20kHz for audi-

tory. Hence, sending more data than the perceivable limitations of human end-user is not beneficial. Machines have a much higher receptive frequency due to different factors: more straightforward computing tasks, differences in computing hardware, etc. Therefore, an application with IoT-AV multiplexing enabled should consider these differences to maximize the benefits. The decisions could include, but are not limited to:

- How the component streams are allocated?
  - In the same RTP session and separated by distinguished SSRCs.
  - In different RTP sessions and grouped by application's record.
- How is the IoT data used in the multiplexed stream?
  - In case IoT data is multiplex with video: sending IoT data after every frame, sending IoT data only after I-frames, or using a new codec of image/video with IoT integrated.
  - In case IoT data is multiplexed with audio: IoT data encoded in one channel of the audio or IoT data sampled at the same frequency with audio
- What packet-loss mitigating mechanism should be used?

Example - UAVs Operational Data

Unmanned Aerial Vehicle (UAVs) is currently one of the active research areas in the IoT, with a wide application range as seen from the market analysis in [61]. Some examples could be military operation devices, last-mile delivery fleets, low-attitude transportation, and on-demand 5G deployment ... In addition to a wide range of applications that requires various specific sensory data, a UAV is usually equipped with sufficient computing capability. These criteria cover both the rationale for using holoplexing and its hardware prerequisites.

For an RTP implementation, a UAV can join an RTP session to transmit directly recorded AV and IoT data while preserving its communication for command transmission. RTCP should be applied for feedback and control of the recorded/playback data with the CNAMEs to group streams' quality on the same device. The component streams are assigned with different SSRCs within the same RTP session. This behavior requires the Multiple Data Type over One Session design described in RFC8872 [30]. Each UAV acts as an active sender whose receiver is the nearest RTP-enabled server. The communication can be described as multiple unidirectional streams from different drones to a common server. A fleet of UAVs in the same operation can join the same RTP session. There are some benefits provided by having multiple session-multiplexing streams in the same session, such as:

- Reducing the network resources: more UAVs can operate simultaneously or for the same service.

- Compensating the lost data from a drone with its fellow's data.
- Sharing the feedback RTCP to adjust the whole fleet's transmission behavior.

However, as a session is identified via a 5-tuple, a single address:port must be used for the sender(s) in this RTP session. For a single drone, it is obvious to have that single value assigned for that drone. In the case of a drone fleet, that group of drones must share the same address:port. An example implementation would be nominating the address of an IP-enabled drone as the fleet's 5-tuple value. The chosen drone will also do the RTCP distributing/processing for all participating drones. Once the session is negotiated, and the transmission is started, the receiver must keep track of the association between multiplexed streams and their SSRCs to perform the demultiplexing effectively. This step requires a more powerful processing unit that can be met and scaled on demand at a modern base station. Tesla [62] is a recent example of the big-data approach to improving the autonomous function of their vehicles, starting with diverse data collection. The UAV industry could also adopt a similar technique to speed up their development with these on-site data.

Currently, the additional information is still limited to the metadata of video frames, such as the UAV telemetry, camera configuration, and gimbal status. Due to the relatively small size, this meta-data could be fitted into the RTP header extension as the work done for Parrot's drone photo metadata embedding [63]. However, as the amount of data to be transmitted increases, an IoT-AV multiplexing solution is necessary for this specific application.

Example - XR

Extended Reality (XR) is another area where applications can take advantage of IoT-AV multiplexing. XR has been researched extensively and is considered the next killer app. In addition to the AV data, other helpful information could come from a gyroscope, a thermostat, a photo sensor, a magnetometer, and so on. For many XR applications, the device is operated in a relatively small area which mitigates the IP address allocating challenge in the UAV example. This application uses the bidirectional communication between the XR device and the nearest 5G fog data/computing center [64] with RTP functions activated to provide sufficient and adjustable computing power compared to the built-in hardware approach and lower total latency compared to the cloud computing approach. In the XR-to-center direction, the end device is an RTP sender that initiates multiple streams within the same session to the center. In the opposite direction, the server is an RTP sender that streams packets to the XR device. The processes could apply different time resolutions due to the various end-users requirement of the transmitted data. The negotiation for multiple data types being used in these sessions is handled via the offer/answer mechanism. An SSRC identifies each distinguished stream, and the relationship between streams is primarily concerned with the fog center. The southbound multiplexed stream must include

RTP with suitable timestamps and corresponding SSRCs to alleviate the lip-sync function in the XR device.

3.1.4 Redundancy

As discussed in the function overview in Section 2.3.3.3, the more redundant packets (duplicates) are sent, the smaller probability of all of them being lost. This subsection will demonstrate some use cases of *Temporal* and *Spatial Redundancy* in IoT streaming.

For temporal redundancy, the network and device tax are similar to having retransmission as default. The time between the original and the duplicate transmission defines the "protected" period in which packet loss can be recovered using temporal redundancy. A system must compromise with more active time on devices and a longer end-to-end delay to provide a longer protected period. For IoT streaming applications, temporal redundancy should only be used in specific scenarios where the trade-off is acceptable. Some examples could be:

- Some streams are critical, and even the transient loss should be avoided.
- The temporal redundancy can be activated regarding the packet sending/receiving reception or application's requirement.
- The resource consumption by temporal redundancy is less than retransmission.
- NAT/ICE/STUN function or multi-sessions management is limited on some devices.

Spatial redundancy's robustness is based on distributing the redundant packets over different sessions. By default, the protected period in this approach is only the moment when the original and its replicate are sent. Therefore, the delay in terms of transmission is not significant, and an end device can make all of its sending before going to low-power mode if needed. However, spatial redundancy requires multi-session management, which could be expensive for constrained device categories. The address-port usage is also higher, followed by a remarkable amount of resources for NAT/ICE/STUN function maintenance. Some common characteristics of spatial redundancy applicable system could be listed as:

- The streaming content is time critical and is worth the network resource extra consumption.
- Participants are capable of joining multi-sessions.
- The buffered replicated packets can be processed with a minimal delay.

Using temporal and spatial redundancy as unit blocks, a system could obtain the *Temporal-Spatial Redundancy* functionality which can significantly improve the robustness of the transmission. As a consequence, this hybrid approach inherits challenges from its components along with significant complexity in both sender and receiver. Given the precon-

dition of effectively re-configuring a session which means it could be using SIP re-INVITE or SIP UPDATE [65] requests, a stream can apply mentioned solutions flexibly to obtain the *Adaptive Redundancy*. Ideally, the redundancy is activated during or right before the lossy period to reduce both the packet loss and the network consumption. The cognitive function of the 5G network[66] could be another source to provide network-level insights for applications to improve the prediction of traffic conditions.

Example - Smart Factory

Industrial IoT (IoT) is an important section of IoT research invested heavily due to its commercial potential. In the picture of Industry 4.0, Smart Factory is considered the next evolutionary manufacturing step. The enhancement is given by smart factories such as[67]: improved process efficiency, lower operational cost, higher product quality, reduced operational accidents, and more adjustable system. A strong correlation between the physical entities and their virtual models in Cyber-Physical System (CPS) is required to achieve the mentioned enhancement. IoT streaming using RTP supports smart factory applications via high temporal resolution and diverse data transmission.

The factory is usually facilitated with power access and a dedicated network that partly alleviates the well-known IoT constraints and allows redundant transmission. In RTP terms, the system's components can be described as:

- End Device is equipped with RTP/RTCP for data transferring and SIP for signaling.
- SIP Server with sufficient coverage and power to handle all signaling from all plant devices.
- Digital Twin Server receives all possible RTP streams from the factory.
- Analytic Sink stores all the operational data from the plant for later offline analytics or simulation.

An assembling line composed of multi-functional robot arms, conveyors, and additional sensory devices as End Devices is used to illustrate a concrete example. RTP streams are set up for data transmission: between End Devices, from the end device to the Digital Twin Server, or from the Digital Twin Server to the Analytic Sink. The first setup is for the collaborative tasks between End Devices, so the transmitted data should be narrowed down to the control-related data, e.g., robot coordination, conveyor speed, or stiffness of the object. Each End Device in the production line receives the list of its collaborators' IDs along with the program to be run during the process. An ID should be illustrative and understandable for humans as the End Device's address:port value is to be received from the SIP Server to assure its validity. The program distributor can be co-located with the SIP Server to take advantage of all devices' IP-address records in the SIP Server. An End Device obtains its collaborators' address:port based on the received ID list and initiates RTP sessions with its collaborators. The implemented RTP topology in this sce-

nario must be Point-To-Point as each participant is a separate physical device with their address:port values. The transmission between End Devices can tolerate a higher redundancy compared with the transmission towards the Digital Twin Server due to the smaller packet size. *Temporal Redundancy*, *Spatial Redundancy* or their variations can be used depending on network capability. However, *Temporal Redundancy* is less desirable in this use case. Because enhancing the "protected" period of a packet means a longer delay between its duplicate and original thus, it compensates for the priority of real-time transmission. As *Spatial Redundancy* is applied, multiple sessions must be initiated and maintained for every pair of End Devices. The Digital Twin Server requires a dual setup for its whole plant model. The transmitted data is of various types and does not contain only related control. Any relative condition of an End Device like temperature, power level, or displacement must be streamed back to the Digital Twin Server to update the Device's model. This transmission can be considered a lower critical level in comparison with the inter-End-Device type if the CPS is only for monitoring purposes. However, a remote operator can order a higher level of redundancy on-demand to serve the critical period or for an online examining task. To activate the additional reliability, the operator needs to initiate more RTP sessions (spatial redundancy) or modify the current RTP session (temporal redundancy) from the SIP Server. The Digital Twin of the factory can prioritize and apply different redundant levels on End Devices, for example, based on their up-time, next inspection date, network situation, and so on. Even if there is a saturation level for applying for redundancy and the number of End Devices in a factory is deterministic, the Digital Twin Server and the SIP Server must maintain tight coordination to ensure the network can afford the redundancy. The last setup is for analytics and storage purposes, so it is not as time-critical. Therefore, reliable functionality, such as retransmission should be considered to reduce network stress.

3.1.5 Simulcast

The technical problem aimed to solve by simulcast is multiple data format broadcasting. The advantages of simulcasting can be listed as follows:

- Devices with different resource constraints and data requirements can operate in the same network. Hence, simulcast strongly supports the heterogeneous nature of IoT.
- The computing complexity and resource consumption are shifted to the stream senders and mixers.
- Less signaling is required in comparison to scalable encoding.

With fewer constraints for the data representations, IoT streaming has more freedom to tailor the content for each use case. In other words, before implementing IoT stream simulcasting, a system designer must define the different "resolutions" for IoT streams

in his application. The data representation selection heavily impacts the performance of the simulcasting function. More details on this topic will be discussed in subsection Data Type of Section 3.1.

The technical details illustrated in Section 2.3.3.4 suggest a simulcast-enabled system requires: a sender with the capability to generate multiple streams of different formats and an intermediary(RTP mixer) to forward the compatible streams to receivers. Because the complexity and resources of a system are mostly focused on the senders and the mixers, such a system can take advantage of the decentralized and cloud-fog architecture mentioned in [64], especially for southbound streaming.

Example - Automotive communication

Vehicle-to-Everything or V2X is an intersection of automotive technology and IoT where a vehicle exchanges data with other entities to extend its application. Various potential use cases and benefits of V2X communications were investigated in [68] and [69], which can be grouped as:

- Safety: Services focus on reducing traffic accidents and minimizing other risks for traffic participants.
- Efficiency: Services focus on optimizing the potentials of both public infrastructure and private vehicles without compromising environmental quality
- Infotainment: Services focus on extra functions for the human user of a vehicle.

Of the three mentioned types of service, the need for stream-liked data transmission persists. In addition, due to the mobility nature of the vehicle and the diversity of things it interacts with, the need for multiple representations of the same information is evident. Examples of such scenarios can be depicted as follows:

> The control station broadcasts the weather or accident notification for multiple autonomous cars (vehicle-infrastructure communication) in an area. The information is given in multiple languages and simulcasted to all vehicles. Vehicles that received the multi-language information can choose to either display just one representation (private vehicle) or multiple representations (public vehicle like a bus)
>
> A vehicle needs to broadcast its condition to other nearby vehicles for safety services. Those vehicles are from manufacturers who may use different representations, such as Fahrenheit instead of Celsius degree or BSON instead of JSON. A vehicle may choose to simulcast data in many popular representations to ensure all vehicles are aware of its condition.

From the aforementioned requirements, simulcast must be considered for V2X communication.

The first example can be described in more detail using RTP terms as the following:

Control Station is an RTP source that generates multiple RTP streams of different formats from the same content and simulcasts these streams to the Mixer.

Mixer is similar to an RTP switching mixer and can be located in a vehicle (a car, a bus, a ferry ...) or a part of the infrastructure (a traffic post, a lighting post, a bus stop...).

Receiver corresponds to the RTP participants and can be an RTP module added to the vehicles. A receiver can serve the received information to a single human user (private car panel or an attached XR headset) or multiple human users (bus' display panel). Similar information can also be fed to the vehicle, but in a much higher resolution(temporal or measurement precision), to improve its autonomous function.

The following procedure illustrates how the simulcast function operates in an accident updating system for automobiles. Each vehicle (Receiver) identifies itself and updates its IP address when entering the coverage of a Control Station. In case an incident happens, the responsible Control Station for that area generates RTP streams related to the incident. The content is expected to be used not only for human users but also for vehicles. Therefore, there are two RTP sessions for each vehicle that have the same IP address but different ports. The human-consuming RTP session is used to transmit data at a low frequency and is only used for presenting insights to humans such as real-time graphs, statistics values, safety recommendations, or audit reports. The vehicle-consuming RTP session contains simulcasted RTP streams that can be used by the vehicle to optimize its function. This information includes:

- Road conditions such as illumination detected obstacles or holes, or icy road
- Traffic flow-related information like congestion detection, traffic redirecting, recovering time estimation
- Public reinforced commands from Control Station such as speed limit or lighting power.

Each simulcasted stream between the Control Station and the vehicle is identified by an SSRC within the scope of an RTP session. A vehicle plays the role of an RTP switching mixer and handles two important functions: updating the data representation it can use and selecting the suitable stream(s) based on that representation. Data from the human-consuming sessions can be displayed directly via the vehicle's info dashboard or speaker. An example would be showing a multi-subtitle graph/map or multi-language audio report. Alternatively, the same information can be forwarded to private devices that are connected to the vehicle. Examples of private devices could be XR headsets, smartphones, or smartwatches. When the vehicle is out of the impact zone of the incident, it can terminate the sessions and prepare to initiate new sessions with the next Control Station.

3.1.6 Multicast

Multicast is required for IoT applications that quickly distribute the same data to many receivers. RTP multicast is desirable for IoT because of the following reasons:

- Lower network resource consumption (addresses, ports, bandwidth).
- A more energy-efficient system as the data is only sent to the needed receivers.
- An abstraction to handle multiple numeric addresses as one entity.
- A shared awareness of data transmission thanks to RTCP feedback reports.
- Possibility to integrate a third-party monitoring application by sharing RTCP feedback.

Within the two groups of multicast available in RTP, SSM should be preferable for its lower complexity and resource consumption than ASM.

Example - Multiple diagnostics service

For medical reasons, a swift reaction during the first hour is critical for recovery when an accident or health-related failure happens. This period is usually called the "golden hour" as its information can make a life-death difference. IoT is already in a fast pace of adoption for healthcare applications such as emergency services as in [70], or early stroke detection as in [71]. Independent of the details on how the patient data is used, a set of high temporal-resolution data is needed by many receivers. Examples of parties interested in this data could be diagnostic services, the nearest ambulance or hospital, the patient's private doctor, etc. IoT streaming with simulcast is a promising candidate to provide this critical data in time.

In this example, an RTP implementation to stream comprehensive vital data (temperature, heart rate variability, ECG, EMG, oxygen level...) to various responsible services. Using the terminologies agreed in Section 2.3.3.1, an IoT streaming system with a multicast function can be realized as:

Original Sender is a sensory device located in a first aid station, embedded in a wearable device, or integrated with an automated external defibrillator (AED).

Distribution Source is assigned as a module at a base station near the Original Sender to shorten the transmission time.

Feedback Target is another module handled at the base station to offload the feedback function from the Original Sender.

Receivers are applications used by services or people pre-assigned for the incident on that site.

Only the Feedback Target is specialized for the RTCP function of the four realized compo-

nents. The other three must be capable of performing both signaling, RTP (sending and receiving), and RTCP functions. Consider the case of an incident, the Original Sender is in its operation mode, and the following tasks are enabled:

1. Collecting vital signals to store, locally visualize and prepare for streaming.
2. Initiating a unicast RTP session to the Distribution Source.
3. Streaming, the signals can be multiplexed into one stream or separated as multiple streams with distinguished SSRCs.
4. Adjusting the streaming process according to the RTCP feedback received from the Distribution Source

Using RTP, all recorded signals are already timestamped and sequenced for reconstruction at the receiving site. After identifying the Original Sender, Distribution Source initiates multiple RTP sessions to prepare for the multicast process. The local operator can register the Original Sender to provide its precise location or send a GPS value along with the SIP messages. When the RTP sessions are ready, the Distribution Source forwards the unicast streams to its Receivers. The receiving services can be dynamically registered to the Distribution Source. Local operators can easily modify the service using the same hardware. For example, if a company is notified about a new employee with a rare allergy, that company can add a diagnostic package specified to that allergy to its current subscription. In another scenario, an ambulance operating near that area can update the patient's situation in real-time to prioritize the action when the team arrives at the site. On a Receiver's side, after joining the session and receiving the data, it sends the RTCP feedback and control signal to a corresponding Feedback Target. The Feedback Target, while allocated in a different physical device away from the Distribution Source, must maintain strong communication with the Distribution Source.

3.1.7 Pause and Resume

As RTP/RTCP is involved in both the application and transport layers, the pause-resume function of a streaming system posed a coherent effect on both layers. Applications benefit from abstraction observed on the application level for pause/resume behavior while still being able to interfere with lower-level operations if needed. Some examples of these benefits can be listed as follows:

- Reducing the number of total operations required for pause/resume behavior. The alternative would be a sequence of terminating the current RTP session and negotiating again a new RTP session, which is more complex.
- Reducing the pause-to-transmission time: If the pause decision is from the sender, it can continue sending at any time. If the decision is from the receiver, that receiver will get the sender's response to its request after a short period.

- Reducing network setup overhead.
- Reusing the available RTCP statistics to better anticipate the transmission of the paused stream.
- Providing a means to indicate the proactively not-transmitting decision (via PAUSED and REFUSED messages) from a sender. This also helps to improve the packet-loss vs. no-packet-sent scenario distinction.
- Reducing the bandwidth usage during the pause period as no additional packet is sent.

In the context of IoT streaming, especially in massive IoT, an enormous number of participants in an RTP session emphasizes, even more, the session's reusability. At the same time, the predictability of the transmission is useful for resource allocation and a short pause-to-transmission time is also desirable in time-critical streaming applications.

Example - Large scale WSN

Examples of pause-resume-enabled systems can be seen in large-scale WSNs with a decentralized architecture. There are cases where high temporal resolution data from all sensory nodes is not desirable, for example:

- The network is temporally allowed for a narrow bandwidth.
- The current task requires only a subset of sensory streams, and the application would prefer to dedicate more bandwidth to those streams.
- Some sensory nodes would like to preserve their battery level to stream only critical data.

In this section, the example use case applies point-to-point topology in a single RTP session. A WSN's components in RTP terminologies can be realized as follows:

Collector acts as an RTP mixer-sender and receives streamed data from sensory nodes in the network. Collector also has signaling capability to receive session configuration from the Controller. Multiple streams from the Collector are within the same RTP session and separated via different SSRCs.

Receiver is a module in a server with RTP/RTCP function. It receives RTP and RTCP streams from the Collector. A Receiver then separates data and control streams before forwarding the former to the RTP Module and the latter to the RTCP Module of the Application.

Application contains application-specific logic and two extra parts: RTP Module for the data ingesting task and RTCP Module for the controlling task.

Controller is separated from the Application and has the signaling capability to arrange the session configuration requested from the Application.

To start with, an RTP session exists between the Collector and the Receiver. The network-aware Application detects a possible bandwidth shortage and reduces its sensory data streams to avoid network saturation. Based on its internal logic and the RTCP reports from the RTCP module, the Application selects a subset of streams to stay streaming. A list of to-be-paused streams is made and sent to the Controller, where it is used to make the required SIP messages. These SIP messages are then sent to the Controller, whose first action is pausing all the indicated streams. If the Application wants to resume a stream, it initiates a new request to the Controller. The Controller then sends a SIP message containing session modification to the Collector. After receiving the SIP message for session modifying, the Collector starts pausing or resuming specified streams. Finally, the Application is informed if any stream is refused to be resumed.

One of the essential benefits of pause-resume is the immediate availability of streams. Because the SSRC space (32-bit) is significantly larger than the number of ports available at an address, the number of maximum simultaneous paused streams depends on the latter range. However, given the same number of ports, the Collector can still expand its coverage further using the CSRC field. Multiple correlating sensory nodes can be realized as contributing sources(CSRCs) for one synchronizing source(SSRC). With this approach, data from contributing sources of the same SSRC is shared fate as the CSRC-level stream control is not yet available. Alternatively, the Controller can initiate a new RTP session to increase the number of streams. This approach is more intuitive but requires early planning for new session negotiations.

3.1.8 Retransmission

Similar to conventional AV streaming, retransmission contributes to the additional reliability of IoT streaming using RTP. Other advantages of retransmission-enabled IoT streaming can be listed as follows:

- Retransmission is more dynamic than Redundancy. The receiver can decide if it wants to have the lost/damaged packet resent.
- Retransmission requires less bandwidth consumption for transient loss than Temporal Redundancy.
- Retransmission consumes fewer transport ports than Spatial Redundancy.

A system designer should also consider the following characteristics of the RTP retransmission to implement that function for IoT streaming:

- IoT devices must be equipped with RTCP to communicate retransmitting signaling. RTCP statistics should also be employed to gain insights into the expected packet arrival time where it is applicable to make the most out of the additional RTCP capability.

- The retransmitted streams can be placed in the same session as the original stream to reduce the number of sessions.
- One should consider tuning the retransmission for practical implementation. A receiver can stop requesting/waiting for repairing packets after several requesting attempts or a time-out. This decision significantly affects the trade-off between reliability and system performance.

Example - Predictive Retransmission Service

Packet loss is a common problem in IoT streaming applications that involve device mobility or sudden environmental changes. The causal factors could be due to blocking by terrain, high level of interference at a particular area, low penetration due to vegetation roof. However, the low transmission quality could be predicted and planned for some of these cases. For the automotive industry, a smart vehicle knows the high packet loss areas when its user plans his trip. That vehicle can then use the position information (GPS) to estimate when retransmission is needed. An additional RTP session is negotiated for retransmission before that car enters the poor connection area. The retransmission implementation follows the session-multiplexing specified in RFC4588. The failed packets are to be buffered in the vehicle as the duration of packet loss is anticipated. The buffered packets are retransmitted until the buffer is empty or the timeout period for those packets has passed. After that, the retransmission RTP session is torn down. The collaboration between network operators and users can provide this predictive retransmission service. The network providers can provide customized based stations to offer a better connection for the vehicle. Organizers and end-users can equip their cars with the RTP-enabled application and subscribe to an automotive data package to benefit from the exclusive streaming quality.

3.1.9 Payload Type

Considering the formats of IoT Streaming, most of the data being exchanged is in the key-value representations such as JSON, BSON, CBOR, Msgpack, Protobuf, or SenML as found in this survey[72]. Compact options like CBOR, Protobuf, and BSON are encoded to binary. On the other hand, JSON and the JSON representation on SenML are encoded in string format like UTF-8. The second group allows RTP infrastructure to quickly adapt to the IoT streaming new use cases by using a text-related extension like "RTP Payload for Text Conversation"-RTP T140. Figure 3.1 shows the format of a T140 packet without redundancy.

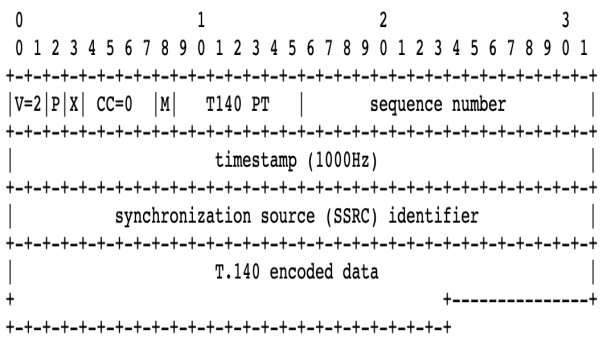

Figure 3.1. *RTP T140 packet format*

By applying encapsulation, a data point or multiple data points can be represented in JSON or JSON-SenML object which can be encoded as a string in the T140 packet. This approach is demonstrated in Chapter 4-Prototype.

3.2 Using SIP

By replacing a human with a non-human in the position of the consumer or an *End User* (EU), VoLTE can be reused to have the data transmission between two machines (M2M call) on that same infrastructure for human voice service. Below is the rationale for the M2M call approach from a SIP perspective.

As a signal protocol for multimedia sessions, SIP supports the streaming part of IoT Streaming as long as a session can be set. The IoT part in IoT Streaming could be covered, too, if an EU can be a non-human entity. One could refer to an entity with the SIP functionalities as a SIP *Endpoint* to emphasize this extension in terminology. The definition of a *Session* in RFC3261 is "an exchange of data between an association and participants" and does not imply that multimedia is the only type of data on the traffic. In this section, *Machine-to-Machine session* or M2M-session will refer to a SIP session with all non-human participants. An M2M session is expected to involve some new behaviors due to the vast amount of differences in EU characteristics. Some examples can be listed:

- The range of session duration significantly varies. A session duration could be extremely short due to the high data consuming/producing frequency compared with human EU. On the other hand, a session of low-power devices could be much longer to prevent the session negotiation.
- A session could contain a significant number of non-human EUs, but each consumes/produces few data than the human EU.
- A session could face a machine-speed joining/leaving condition from many EUs, which may require inter-session functions.

Alternatively, a SIP session could also be a *Person-to-Machine session* or *Machine-to-Person session* if the EUs include humans and machines.

It can be considered that, in the absence of major changes in SIP, a broader implementation of sessions between a diverse population of IoT EUs is possible. There already exist commercial platforms to provide SIP Servers as a Service for IoT applications like alarm servers (iotcomms.io [73]) or automated parking(SIPazon [74]). The following subsections describe the novel use cases for SIP in IoT Streaming.

3.2.1 IoT PubSub system using SIP conference

3.2.1.1 Rationale

By applying solutions from the PubSub paradigm, multiple inherited challenges from both IoT and Streaming areas in some cases could be dealt with. The common advantages of telecommunication infrastructure reusing have been discussed; hence, it is not mentioned again. Other immediately recognizable benefits can be listed as follows:

Providing one-to-many data distribution

Decoupled communication: Data transmission does not require Publisher and Subscriber activeness simultaneously.

Supporting high temporal resolution data transmission if needed.

Event-driven: no polling is required, and the data is sent whenever an event is observed/produced.

Energy Efficiency: End-devices can concurrently execute their communicating function quickly and then return to less resource-demanding tasks (or sleep mode). The broker handles the resource-demanding task of topic bookkeeping, routing, and policy management.

Modularity and Scalability: The abstraction of broker(s), modularity, and extensibility from the core principles of PubSub provides interoperability as a default to deal with the heterogeneity of IoT. The broker can be scaled dynamically and horizontally in advance, increasing the potential of PubSub systems.

Bandwidth saving: The massive number of IoT devices can quickly saturate the network's traffic if the real-time data is delivered using the Request/Response model.

The correlation in their designs can be realized with the analysis given in Section 2.6-Conference and Section 2.7-PubSub. Both technologies introduce the separation in functions: Signaling vs Data Transferring for Conferences and Topic publishing/subscribing vs data forwarding. Both of them are capable of one-to-many transmission and streaming tasks. A centralized server is required for a PubSub system and most conference archi-

tectures. This correlation encourages an attempt to re-purpose the Conference using the SIPPING framework to set up a PubSub system.

3.2.1.2 Components

From RFC4353, there are elements in the framework for conferencing that can be used to implement PubSub behaviors for a communication system. The essential components of such a PubSub system are listed as follows:

- Participants in a SIP conference are Publishers or Subscribers.
- A SIP conference is assigned with a topic in PubSub. In an implementation, a topic is associated with a Conference Server composed of: A Focus, A Conference Notification Service, A Conference Policy Service, and Mixers if the mixing function is located at that Conference Server. A Conference Server only handles the operations regarding the topic associated with it.
- A Conference Factory in SIP conferencing acts as a broker in PubSub. It is an entity that can allocate resources for multiple conferences and create/modify those.
- Other intermediate components like middlewares are context-dependent, so they are not described in detail here.

3.2.1.3 Storing PubSub information in a Conference Object

The realized components require means to persist the related information about a PubSub system like the Publisher list or the address of the Publisher. This information can be stored in the Conference Object in the Focus of a Conference Server. Conference Object was defined in RFC4575-A Session Initiation Protocol(SIP)Event Package for Conference State[56]. This subsection describes the findings about reusing these fields for PubSub. The primary idea behind this subsection is to reuse the available event package in AV conferences for IoT use cases as much as possible. If there is no suitable field to store the valuable information, this subsection suggests some possible solutions.

`<subject>`

The name of a topic can be placed in the `<subject>` field within a `<conference-description>` of a conference. Initially, in a conventional AV conference, `<subject>` is used for descriptive usage for human users. Therefore, it could be more beneficial to reuse this element rather than adding a new one to the package. To maintain the uniqueness of a topic name, the Conference Factory is assigned an additional task, topic collision avoidance.

`<keywords>`

Different values in the `<keywords>` field in `<conference-description>` are helpful for the Conference Factory to filter conferences it created. In a PubSub realization, it is equal

to the ability of a broker to filter its topics.

`<roles>`

The Conference Object contains a field `<roles>` to indicate the allowed operations for a participant swiftly. Participants in a PubSub implementing system can use this field to express their role as subscribers, publishers, or both. While field values like `publisher` or `subscriber` can directly describe the role of a participant in PubSub, a system designer should also consider reusing the existing SDP attributes for this purpose to avoid the additional mapping task. For example, a publisher can be described as `sendonly`, and a subscriber can be described as `recvonly`. If this solution is applied, the role of a participant can be copied from SIP messages to the Conference Object without conversion.

`<available-media>`

The successfully negotiated RTP stream characteristics for a topic can be stored in the `<available-media>` field in the `<conference-description>`. Multiple entries can be saved in this field, making it convenient for multiplexing or simulcasting-enabled applications. The sub-field `<media>` allows audio, video, text, application, and message. The "text" and "application" formats are promising to describe the IoT data stream.

`<status>`

The `<status>` field of an `<endpoint>` is a handful to store the current condition of a PubSub participant. For example, an Endpoint with a "connected" status is ready to receive notifications from the topic and the data forwarding from the corresponding mixer. An Endpoint with a "pending" status could be paused and refuse to receive the data while still receiving PubSub notifications.

In addition to the field mentioned above, other enhancements can be added to the SIP and SDP to support PubSub operations better.

`<subscriber-count>` and `<publisher-count>`

Quick access to the number of publishers or subscribers often benefits policy-based behaviors. For example, if a topic has had no subscriber for a while, the Conference Server can stop the streaming from its publishers to save battery for those devices. This adjustment provides more actionable information than the general `<user-count>` given by the SIP Conference Event Package.

`<status>*`

Negotiating the RTP stream characteristics every time a participant re-joins a topic could be expensive for IoT devices, as discussed in the *Pause/Resume* Section. It is valuable to have a value in the `<status>` field of `<endpoint>` to indicate the "paused" status of

an Endpoint explicitly.

3.2.1.4 SDP attributes for PubSub

As SDP is used for media negotiation for RTP sessions. The IoT PubSub realization also requires a similar describing function for IoT data session negotiation. Hence, it is valuable to have an attribute dedicated to storing the PubSub topic associated with a conference. An example of this new attribute could be ‘a=topic:<topic-name>‘.

Regarding the corresponding Conference URI of a topic, it can be placed in the u=<uri> attribute of the session description. This attribute is optional in the SDP format and only used to describe additional human-readable information. However, each session can have only one u=<uri> attribute. Therefore, one should come up with another solution if multiple conference URIs for a conference is the case. One example scenario would be when a conference shares different URIs for participants of different roles. For such requirements, an additional attribute for conference URI is desired to store URIs in the media description part of the SDP. An example of a SIP message containing a conference URI can be seen in Listing 3.1. The topic of "temperature" is associated with a conference whose conference URI is "sendonly-temperature@server:port". Participants can only publish data using that conference URI due to the a=sendonly attribute.

Listing 3.1. *Conference URI in a SIP message example*

```
...
a=topic:temperature
a=confUri:sendonly-temperature@server:port
a=sendonly
...
```

3.2.1.5 Topic-Conference URI mapping

For a participant to join any conference, the conference URI of that conference is required. While the association between a topic and the conference URI exists in the conference object in the Focus of the Conference Server, participants have no access to that information. However, a participant in a PubSub implementing system has the name of the topic it wants to interact with. Therefore, a new entity needs to be defined whose primary function is maintaining a map for topics and their corresponding conference URIs. In this thesis, this entity will be referred to as Topic-ConferenceURI Mapper or Mapper, for short. A Mapper can operate as a network function who are reachable by any components in PubSub systems that need to convert between a topic and a conference URI.

3.2.1.6 PubSub operations

To create a topic

A topic in this realization is created after the Conference Factory successfully creates a conference with the `<subject>` value of that topic name. Hence, there are at least two cases that lead to a topic creation:

- A publisher or subscriber creates a topic by sending a SIP message to the broker. The simple flow can be seen in Figure 3.2.
- The broker receives the command to create a topic from another application logic. Then, the broker creates a topic without any participants.

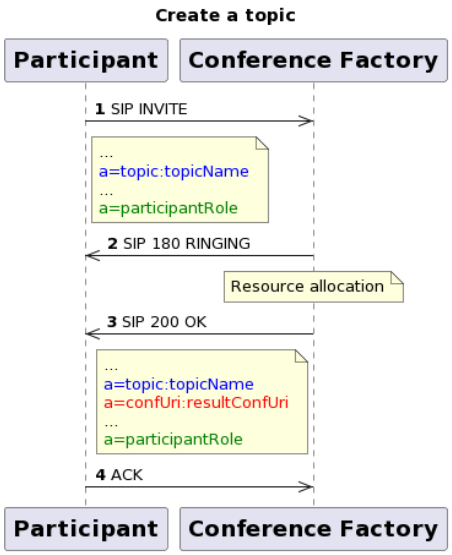

Figure 3.2. *Topic creation by a participant*

If the broker receives a request to create an already existing topic, i.e. the Conference Factory is asked to create a duplicate conference, some options for the broker's behaviors can be:

- Similar to the AV conference: The Conference Server sends a SIP message (4xx-Client Failure Response) back to the request originator. The follow-up behaviors, like trying again or modifying the SIP message, depending on the participant's policy.
- More IoT-oriented: The Conference Server adds that participant to the existing conference instead of sending back the error response. This policy is favorable for PubSub systems containing numerous constraint devices. Fewer SIP messages are needed, and the participants can start sending or receiving data earlier.

To publish/subscribe to a topic

A participant can try to interact with a topic as a publisher or subscriber using the corresponding conference URI for that specific purpose. Participants communicate with a Conference Server instead of the Conference Factory as in the topic-creating case. The interactions with a topic can be initiated in different scenarios, for example:

- A participant makes the initiation of the interaction itself. This is equal to a publisher or a subscriber sending a SIP INVITE message to the Conference Server to invite itself to that conference.
- A participant is referred to a topic. It means another in-topic participant sends a SIP REFER message to the Conference Server to introduce a new participant to that topic.
- Application logic in the Conference Server, after receiving commands somewhere else, adds participants to the topic. The Conference Server sends SIP INVITE messages to different participants and waits for their responses on whether to join that conference.

If participants already have the conference URI, the call flows are similar to the AV conference call flows. However, additional flows are needed to acquire the conference URI from the Mapper if a participant only has the topic value. Figure 3.3 shows an example of this extended call flow. The protocol used for those added flows is unspecified, as it could be tailored for different use cases.

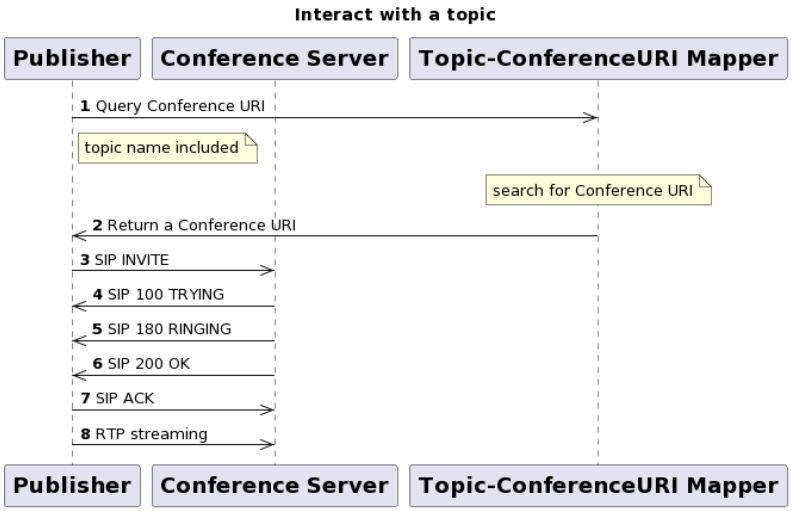

Figure 3.3. *Example call flow of a publisher publish data to a topic*

To destroy a topic

A topic can be destroyed similarly to how a conference is destroyed, as mentioned in RFC4353:

> By a broker: If the broker decides to destroy a topic or if other causes destroy it and

a broker detects that, that broker must inform the topic's publisher and subscribers to stop their PubSub-tasks.

By a publisher: A publisher can stop streaming its data to a topic. To operate correctly, the publisher must inform the broker about its stopping. The broker, after that, signals to all subscribers that this topic is about to shut down, and they should stop waiting for streaming data.

By policy: A Conference Server operating the topic can also be equipped with the ability to destroy a topic itself. An example use case would be a topic with expired time preset by a service provider. If the expired time is met and no extension has been received, the Conference Policy Server can ask the Conference Server to start the conference destroying process.

3.2.1.7 A new SIP PubSub Event Package

Suppose the usage of IoT PubSub over-expanded the capability given by the reusing approach mentioned in the above subsections. In that case, defining a new event package for PubSub operations using SIP is more beneficial and practical. Using a tailored event package would reduce the confusion caused by the function mapping between AV and IoT data conferences. It provides a more unified and efficient solution for IoT PubSub-specific use cases. On the other hand, defining a brand-new event package requires significant time and resources to structure such a systematic and comprehensive solution.

3.2.2 System recording

3.2.2.1 Rationale

The importance of recording or data capturing in this era can be considered critical to mandatory in many cases. The RFC6341 [75] provided a broad list of use cases of recording for media streaming which can be useful to realize similar applications for IoT Streaming. Table 3.1 sums up those use cases with a more general scope of the type of data being streamed. RFC6431 also laid out some terminologies widely used in the follow-up document about the recording function:

SRS -Session Recording Server is a SIP UA that acts as the sink of recorded data.

SRC -Session Recording Client is a SIP UA where the recorded data is sent from, towards the SRS.

Two distinguished types of Sessions: Communication Session(CS) as the subject of the recording and Recording Session(RS) to record that CS.

Metadata to describe the CS and its recorded data.

Usecase name	Description
Fulltime Recording	One RS for Each CS
Selective Recording	Apply recording only for a subset of CSs
RS within a CS	Start/Stop RS during the active time of a CS
Persistent Recording	A single RS is run continuously for one or more CSs
Realtime Recording Controls	A portion of streaming data in a CS can be demanded not to be recorded
Voice Portal Recording	Specified for voice involved application, recorded data is used to improve performance or to adapt with compliance requirements
Enterprise Mobility Recording	Focus on enterprise usage, to record the work-related data being generated when their employees are not on company premises.
Geographically Distributed Recording	The Recording scope is defined by the location of the SRCs
Complex Scenario Recording	The operation of an RS leads to another associated RS to be done
High Availability Continuous Recording	To support RS with extreme requirements
Multichannel/Multi datatype Session	An RS supports stream data that comes in multiple channels and of different types. This also involves synchronization.
Realtime Processing	A SRS must be able to support real-time processing of the data in an RS if needed

Table 3.1. *Recording Usecases*

In the rising tide of Machine Learning and Big Data applications, the demand for data capturing is even more severe as the huge amount of data is an enabler for those technologies. Therefore, it is evident that the recording function should also be supported once network operators start to provide IoT Streaming on their infrastructure. From the perspective of an individual user, if the recorded data can be secured and shared only with his permission, there are potentials for personalized service adoption and monetizing personal data. Hence, the users could benefit more from data recording and are

encouraged to provide a larger data collection.

3.2.2.2 Findings and Design

Given the IoT Streaming discussed in the previous section of this thesis is enabled, the recording function can be conveniently implemented by initiating a session with a similar setting between SRS and the SRC. In other words, a system with IoT Streaming capability should be able to extend its functionality to cover the recording function, without major changes. Different architectures to support recording are described in more detail in RFC7245 [76].

3.2.2.3 Example

This subsection discusses the use cases of the recording function for remote support in IIoT. To enhance the adoption rate of automation and robotics in the manufacturing section of IIoT, customer support must be considered an essential part of their products. The recording function can be realized in different scenarios for this use case:

1. On-premise-SRS: The recording system can be set up and owned by the factory where devices are operating. In the presence of an incident, the factory can retrieve information about the devices around that time. This data can serve their internal analysis or be selectively cut and sent to the device manufacturer for expert support. This scenario is similar to the Geographically Distributed Recording where the scope of recording is the factory area.
2. Manufacturer-based-SRS: In case of an incident, the on-site engineer can connect to the remote support provided by the manufacturer. During the debugging session, data generated by a faulty device is transmitted to the manufacturer. This session can be further augmented with video and audio from multiple devices, i.e. the recorded interaction from the operator during the debugging process. The manufacturer can request recording permission to improve their services or create a fault profile tailored for that specific factory. The scenario requires both the Complex Scenario Recording and the Multi-datatype Session.
3. Automated-Recording: A new scenario can be realized by combining The Realtime Recording Controls and Realtime Processing use case. The manufacturer can also provide an automated diagnostic service for the factory with the precondition of recording. If an incident happens, the diagnostics service will run multiple popular tests to catch the common causes of failure and record those results. In case it requires more investigation, the process continued as in Manufacturer-based-SRS, but the factory owns the right to stop recording at any time.

3.2.3 Constrained SIP

The HTTP-based design of SIP results in the text-based protocol syntax that is useful to improve interoperability and simplification during the implementation/debugging phase. However, text-based messages also increase the network traffic and resources to store/process. Constrained IoT networks demand a more constrained-device-oriented version of SIP. To resolve the message-size problem of SIP, a fusion between SIP and CoAP [77] - Constrained SIP (CoSIP) [16] was proposed as a solution. The idea is to have a binary protocol whose message format resembles CoAP while reusing the rest of the SIP design (architecture, concepts, header fields...). Figure 3.4 illustrates the approach of CoSIP architecturally, which emphasizes the reusability of SIP. In his follow-up work [78], Simone Cirani presented a reduction of 33 percent of SIP-related data the in an IoT application prototype using CoSIP. The same research also pointed out the potential to reduce the number of received packets at a lower layer where packet fragmentation stresses the network even more.

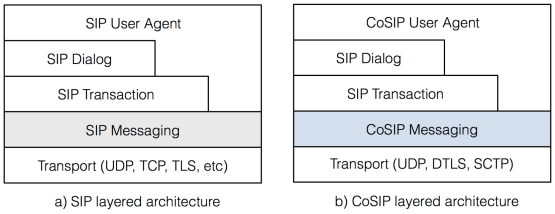

Figure 3.4. *Comparison of the layered architectures of SIP(a) and CoSIP(b). Adopted from [16]*

A lightweight Session Initiation Protocol (Lightweight SIP) was introduced in [17] and proved to provide an even more significant improvement in terms of network traffic, as can be seen from the significant size reduction between the first and third columns in Figure 3.5. While CoSIP only uses the binary message format from CoAP, Lightweight SIP goes further into the CoAP realm to apply CoAP's methods and REST paradigm for session-related tasks. As a result, Lightweight SIP also allows reusing more CoAP and RESTful stacks available in other software libraries.

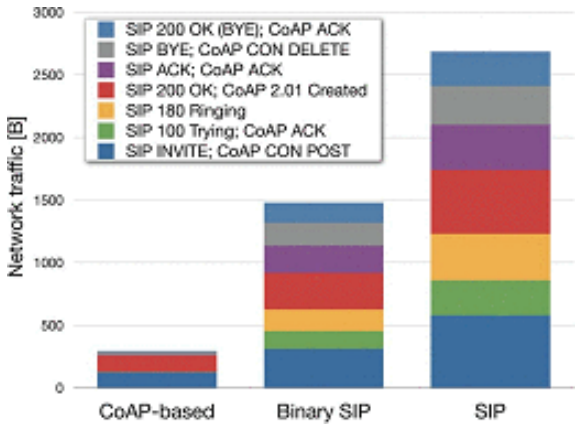

Figure 3.5. *Network traffic comparison between: Lightweight SIP(CoAP-based), CoSIP(Binary SIP) and Standard SIP(SIP). Adopted from [17]*

In conclusion, if these compact SIP archetypes can be standardized and applied to IoT streaming systems, significant network traffic can be saved for devices with limited capability.

3.3 On VoLTE

This section is dedicated to discussing VoLTE infrastructure for IoT Streaming purposes.

3.3.1 Benefits

By reusing the same infrastructure made for the AV streaming services for IoT Streaming, multiple parties involved in the VoLTE/IMS network infrastructure can reach a mutual gain. First, integrating IoT Streaming into the VoLTE network can provide a head-start for those new services and result in stronger competitiveness. The list below contains some advantages of the VoLTE/IMS infra that can serve as prominent traits for IoT Streaming service providers to enhance their user's experience.

- There are numerous completed functionalities ready for deployment. Some important examples could be subscription management, authentication, or recording.
- The physical network infrastructure that covers a wide area(especially residential zones) with a high-quality connection and on-time maintenance services.
- With the adoption of hyper-scale computing in the telecommunication industry, services run on VoLTE can be supported in case of wildly fluctuating data traffic, which is even more valuable for IoT use cases.
- The inter-operator compatibility can be provided and paves the way for roaming services if necessary.
- With a proper abstraction from service providers to hide the heavily technical de-

tails, like by using APIs, the telecommunication functions can be exposed to a larger audience, including Small and Medium Enterprises and individual developers. Strong bonding with the community is valuable in the long run.

Second, IoT Streaming leads to new business opportunities for the same investment. Similar to today's popular mobile data subscription, connectivity for IoT can also be provided in the same manner. The subscription plans are expected to be more dynamic due to the diversity of IoT device classes and applications. The IoT subscription can be considered another stable revenue for service providers. Hardware providers and service providers can also collaborate on a single-purchase package that includes all IoT hardware and the necessary connectivity to operate that hardware. This package can range from wearable devices needing to update health signals to a fleet of industrial robots demanding high-speed control.

The third benefit comes from merging the existing and the new use cases. For example, the same customer who enjoyed the AV streaming on their smart TV is likelier to enjoy the IoT-AV multiplexed streaming service on her XR headset. On the other hand, a factory whose surveillance system is solely based on AV recording can change to an audio and ultra-high sound approach to reduce the cost or prevent sensitive information from leaking. These cases are extremely important for the technology adoption process before the full-blown stage of IoT Streaming.

Lastly, companies with knowledge of VoLTE networks can significantly benefit from their human resource as less investment is required than adopting new IoT Streaming technology. Another advantage could be the community impact and social connection from the previous research, standardizing work, and product development.

3.3.2 Reusing all three domains in a VoLTE network

The same simplified architecture can drive the structure of this discussion in 2.14.

On the Radio Access Domain

The scale of IoT networks would demand a more demanding Radio Access capability to support the increasing traffic. Depending on each specific use case, an IoT Streaming application would require a Radio Access Network with the following:

- A large range of coverage.
- A denser base-station distribution at critical locations, for example, in highly populated areas or stages for public events.
- A capability to expand with a well-defined procedure for it, both technically and legally.
- A regular maintenance and updating plan to keep up with industry standards.

- A sufficient accessing and mobility support.

These bullet points are all available in the Radio Access network that is being used for VoLTE as described in 2.5 Therefore, this domain can be considered the first step in the IoT Streaming adoption process in VoLTE.

On the EPC Domain

From two 3GPP Releases, 13 and 14, LTE-M was introduced along the enhanced Machine Type Communication(eMTC). Compared to its sibling NB-IoT, LTE-M provides higher mobility, a higher data rate, and especially native support for VoLTE. This standardization opens the possibility of repurposing legacy or exploiting under-performed LTE networks for a new application like telemetry and tracking system as LTE shares part of its spectrum with LTE-M. In addition, the LTE-M Deployment Guide [79] includes requirements and suggestions for implementing LTE-M on an Evolved Packet System (EPS). Visibly, the same EPS infrastructure VoLTE could serve M2M communication as a repurposed or additional service. There are related works in both research areas [80], [81], and industry [82] that indicate interest in this approach. Since IoT streaming applications would require 5G connectivity, EPC is expected to stay relevant in the future to share the traffic pressure with the 5G Core.

On the IMS Domain

In the implementation plan of the IMS domain in VoLTE, SIP and RTP/RTCP are chosen as the main protocols for network signaling and data transferring. Therefore, the realizations described for SIP and RTP/RTCP can be inherited for the IMS network. However, the IMS implementation does not contain all possible functionalities specified in RFCs. The IR92[50] is a recommended official document for operators who want to directly apply the Generic IMS Functions or enable the Supplementary Services for their IoT Streaming applications.

3.4 Summary

Based on the analysis of the various technologies explored in this chapter, it was determined that the multiplexing and simulcasting functionalities from RTP are likely to be particularly useful for IoT streaming applications due to their ability to support efficient and reliable transmission of large amounts of data. Additionally, SIP was identified as a critical protocol for facilitating signaling in IoT streaming applications, as it is commonly used in other telecommunications contexts. Finally, the PubSub architecture was selected for implementation in one of the prototypes. It is a popular approach for many IT applications and can support many IoT use cases. The selection of these technologies for the prototype implementation was based on criteria, including their ability to support real-time data streaming, their compatibility with existing protocols and standards, and their potential for

supporting a range of IoT use cases. In the next chapter, these technologies will be implemented and tested in practical use-case scenarios to demonstrate their effectiveness and feasibility for IoT streaming applications.

4. PROTOTYPE IMPLEMENTATION

This chapter demonstrates two prototypes to materialize the concepts and findings from Sections 3 and 4. The first was focused on implementing streaming IoT data using the SIP/RTP stack. Two discussed RTP extensions, namely multiplexing, and simulcasting, were added to the initial signaling/data transferring system. In the second prototype, a PubSub system for IoT devices was implemented using the findings from the PubSub realization part. This section is organized into three parts: Section 4.1 mentions the common approach for both prototypes, Section 4.2 contains all content about the implemented IoT data streaming using RTP (with two extended functionalities), and Section 4.3 is dedicated only for the PubSub operations.

4.1 About the prototyping process

4.1.1 Time Distribution

The most effort during the prototyping phase was placed on implementing signaling/data transferring core functions and defining IoT data format. The additional RTP features and visualization were later added based on the best-effort approach in the IoT streaming prototype. For the IoT PubSub, time was distributed more for the call flow defining and implementing the additional function (Topic-ConferenceURI Mapper)

4.1.2 Feature Expansion

In these prototypes, the software-package modularity was applied to ease the debugging in the development process and assure future feature expansion. For example, suppose a feature like multiplexing was added. In that case, extra processes like multiplexing and demultiplexing should be easily integrated (before and after the packet transmission process) without a major change in the system.

4.1.3 Tech Selection

Both prototypes were developed using Golang(Go) for the following reasons:

- Complete programs are compiled into binary, reducing resource consumption in constrained devices.
- Go is embedded with testing features that help develop multi-agent software.
- Go is adopted in many telecom and cloud-native projects.
- Prototyping using Go is faster than in C or Rust.

This prototype was developed by expanding and contributing to two open-source Go packages: `go-sip-ua` and `pion/rtp`. The `go-sip-ua` package was used to assemble the SIP Proxy Server, SIP Registrar, SIP UAC, and SIP UAS entity. The `pion/rtp` was utilized as the inspiration for implementing a packetizer/depacketizer for the T140 format. This package did not yet support this payload type, so a Go package was made following RFC4103[83] to provide text transmission over RTP.

For data storage purposes, InfluxDB was chosen because of the time-series characteristic of the data and the option to implement databases in both cloud and local host manner. As a result, Grafana was selected for the visualizing tasks thanks to its dashboard support and smooth integration with InfluxDB. Before the final deployment, the application for each component has been containerized using Docker for testing in an internal Docker network.

4.2 IoT streaming

4.2.1 Target

To demonstrate the feasibility of IoT streaming over RTP, a prototype is set up with the following targets:

`Target 1` Deploying signaling and data transferring system on constrained devices using SIP and RTP.

`Target 2` Defining the packet payload format for IoT data using RTP.

`Target 3` Streaming IoT data in the format defined in `Target 2` using the infra from `Target 1`.

`Target 4` Implementing one or more additional RTP functionalities as discussed in Section 2.3.3

4.2.2 Design

4.2.2.1 Minimal Components

The minimal prototype includes three entities as described in Table 4.1

	SIP Server	Stream Sender	Stream Receiver
Composed of	SIP Registrar, SIP Proxy Server	SIP UAC, RTP Sender	SIP UAC, RTP Receiver
Operation plane	Signaling	Signaling and Data transferring	Signaling and Data transferring
Hardware	Laptop or Raspberry Pi 3 (RPi3)	RPi3 with a sensory peripheral (Sense HAT[84])	RPi3

Table 4.1. *IoT Streaming prototype components*

4.2.2.2 Operation Sequences

The operation sequence of the whole system can be seen in Figure 4.1 if both users are registered at the SIP Server.

The prototype assumes the Stream Sender achieved the contact (URI) of the Stream Receiver by other means. The operation sequence consists of four processes:

1. UA Registration: both UAs register themselves as online at SIP Server
2. Stream Initiation:
    - Stream Sender sends an INVITE message to SIP Server.
    - SIP Server uses the contact in that message to check if Stream Receiver is online.
    - In case the SIP Server finds the contact, it forwards the INVITE message to the Stream Receiver to initiate a streaming session.
    - Stream Receiver accepts to start a streaming session and passes its answer to SIP Server, who passes that to the Stream Sender.
3. RTP Streaming: The IoT Streaming process in an RTP session starts.
4. Stream Termination: The Stream Sender stops streaming data and sends a BYE message to indicate that Stream Sender is not sending more data till a new streaming session is initiated.

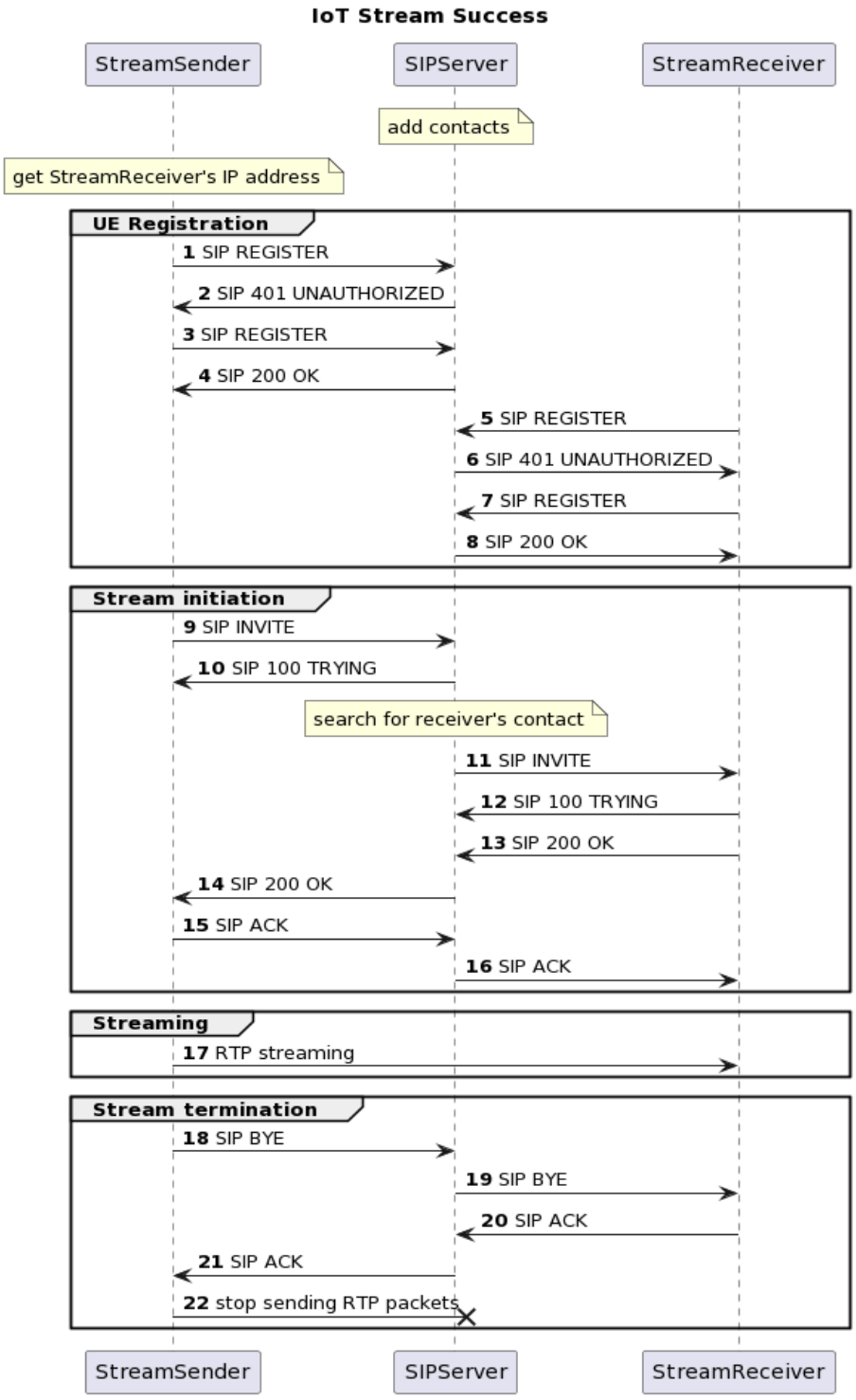

Figure 4.1. *The sequence diagram of a successful IoT streaming process*

To provide more details on the "IoT Streaming" operation in Figure 4.1, the flow of an IoT data stream from the Sense HAT peripheral to the other end of the cloud-hosted database can be seen in Figure 4.2. A packetizer requires a list of parameters, some of which come from a successful session negotiating process. A packetizer is assigned with an SSRC value. Hence it only packetizes data for a single stream.

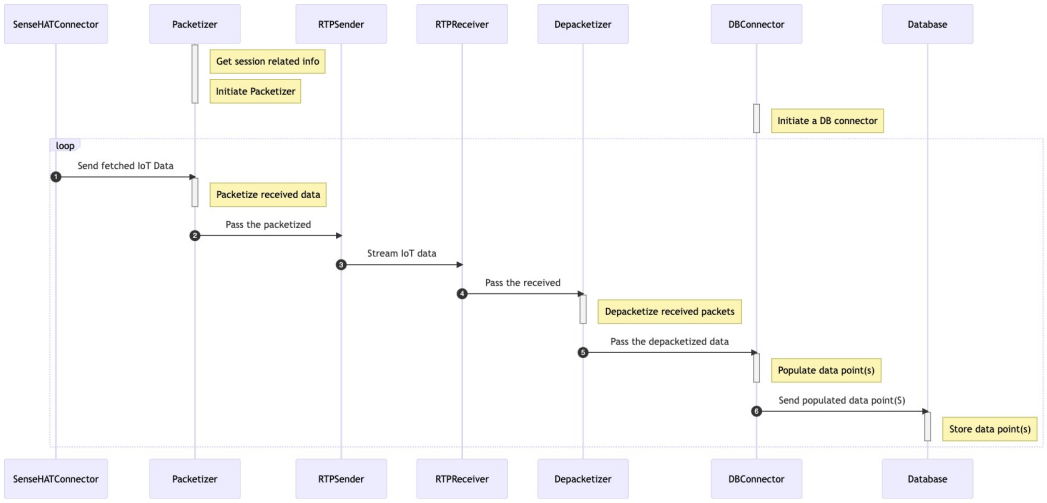

Figure 4.2. *The end-to-end flow of data in IoT Streaming Core Architecture.*

Figure 4.3 illustrates the system operation if the contact of the Stream Receiver used in Stream Sender's INVITE message does not match with any AOR in the Registrar.

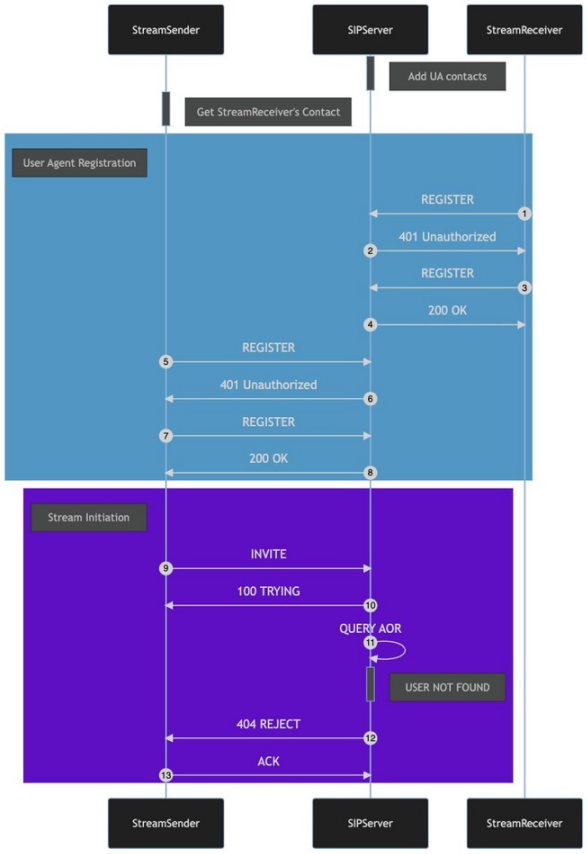

Figure 4.3. *The sequence diagram of a "404 USER NOT FOUND" IoT streaming process*

Only the first two processes are included in the failed streaming process: the UA Registration and the Stream Initiation.

4.2.2.3 Architecture

IoT Steaming Core Architecture

Based on the components and operations described in the sub-section above, the architecture of the minimal system that demonstrates the core signaling/data transferring functions was formed and can be seen in Figure 4.4.

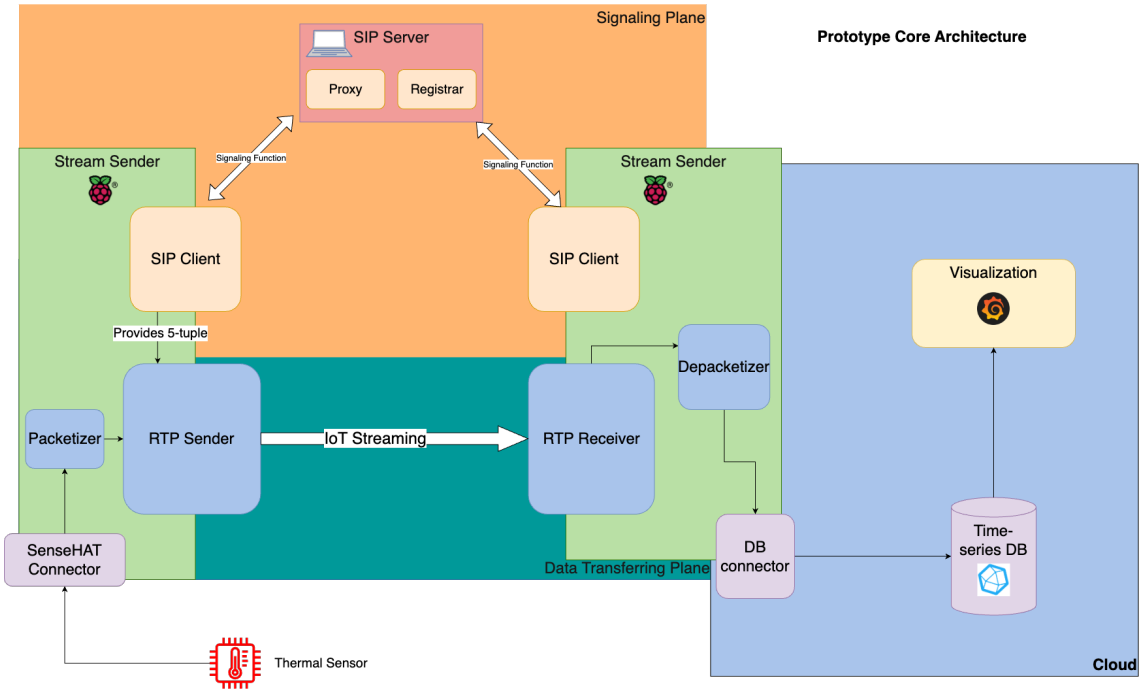

Figure 4.4. *IoT Streaming Prototype Core Architecture*

IoT Streaming Architecture with Multiplexing Support

Additional components have been added to the Core Architecture to support the multiplexing function. In the demo, as shown in Figure 4.5, the multiplexed stream includes the measurements from three sensors measuring temperature, humidity, and pressure. When these three measurements are separated, they are coded in different colors (red for temperature, blue for humidity, and purple for pressure). The multiplexed stream of data is coded in white color.

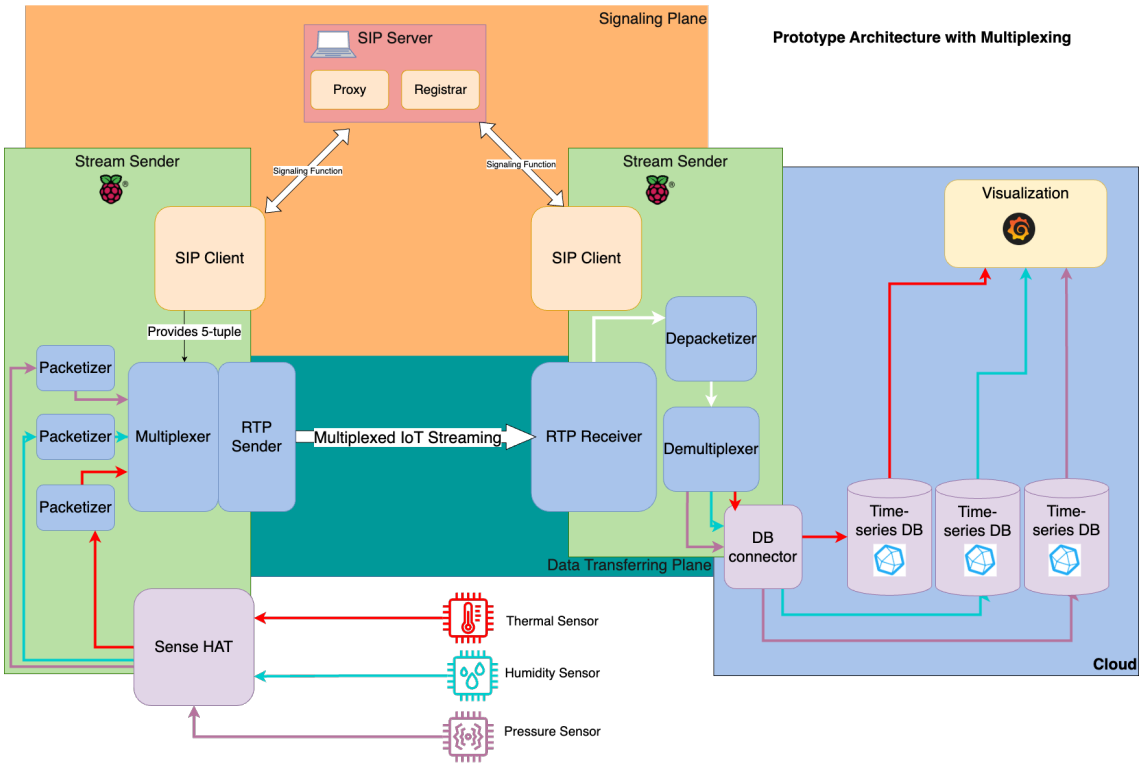

Figure 4.5. *IoT Streaming Prototype Architecture with Multiplexing Function*

The chosen multiplexing scheme was SSRC-multiplexing, meaning packets from the same stream were identified by their SSRC field. Consequently, three packetizers, each with a distinct SSRC, were also required on the Stream Sender side. On the Stream Receiver side, a demultiplexer was added to sort the depacketized data into their original streams. The additional databases were optional.

IoT Streaming Architecture with Simulcasting Support

To demonstrate the simulcast function, the orientation (roll, pitch, and yaw) measured from the sense HAT have been chosen. There were two packetizers on the Stream Sender side: one to packetize data in low resolution (marked with a thin black arrow) and one to packetize data in high resolution. In this prototype, the difference was illustrated using integer type for the low-resolution stream and float type for the high-resolution stream. As a result, each packetizer had a unique SSRC and Payload Type. The "Simulcast Logic" was added to decide how the simulcasted data should be processed. Both streams were sent to different storage in this prototype and visualized in Grafana dashboards. However, Simulcast Logic can be reconfigured with other behavior, for example: visualizing the low-resolution data for human consumers and sending high-resolution data to mission-critical computing nodes. The IoT Streaming Architecture with Simulcasting can be seen in Figure 4.6

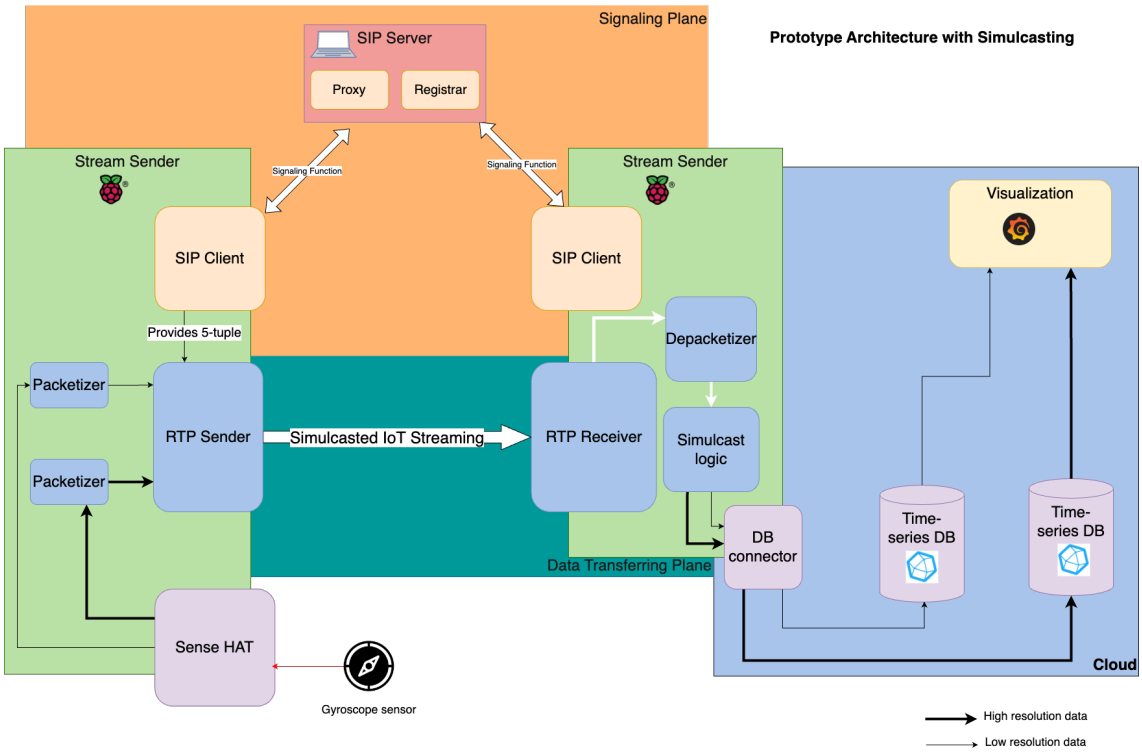

Figure 4.6. *IoT Streaming Prototype Architecture with Simulcasting Function*

4.2.2.4 Payload Type

Each group of measurements from the Sense HAT was organized as a JSON representation of SenML (JSON-SenML), which comes in the form of a list. This list is then serialized and placed inside an RTP T140 payload. An example of an IoT packet is in Figure 4.7

RTP T140 Header

RTP T140 Payload = a serialized SenML list

```
[

{"bn": "demo:raspberry:sensehat/",
"bt": 1276020076,
"bu": "Cel" },
{ "n": "temperature", "t": -2, "v": 25.5 },
{ "n": "humidity", "t": -2,"u":"%RH", "v": 87.5 },
{ "n": "pressure", "t": 0,"u":"hPa", "v": 1023.25 },
...
                                                ]
```

Figure 4.7. *An example of IoT packet used in this prototype*

4.3 IoT PubSub

4.3.1 Target

This prototype was made to prove the applicability of using conference-related technologies in SIP(the conference framework and the conference event package) and RTP for IoT PubSub purposes. The incremental steps towards that goal could be listed as follows:

`Target 1` Reusing the IoT streaming in Section 4.2 as fundamental blocks for IoT PubSub.

`Target 2` Implementing the basic operations of a PubSub system:

- topic creating/destroying
- data publishing/subscribing/unsubscribing
- one-to-many data distribution

`Target 3` Demonstrating the IoT-specific aspects of this prototype like the Topic-ConferenceURI Mapper.

4.3.2 Design

4.3.2.1 Components

In this prototype, all components were deployed as docker containers in the same virtual network due to the limited time budget and the ease of repeatedly developing/testing the entire system. The components are described in the following paragraphs.

Publisher

A Publisher in IoT PubSub is similar to the previous prototype, consisting of two sub-components: a SIPClient for signaling tasks and an RTPSende for data-transferring. If the Publisher also shared Conference URI, an additional sub-component could be required. However, in this demo, this Conference URI sharing was not demonstrated.

Subscriber

For IoT PubSub, a Subscriber was extended with an HTTPClient to enable communication with the Topic-ConferenceURI Mapper, whose name was abbreviated to ConfTopicMapper in this demo. The other two functions were identical to the one in IoT Streaming. A summary of a Publisher/Subscriber function can be seen in Figure 4.8.

Publisher	
SIPClient	to communicate with the ConfFactory and the SIPServer in Server
RTPSender	to send packets to the MRF

Subscriber	
SIPClient	to communicate with the SIPServer in Server
RTPReceiver	to receive packets forwaded from the MRF
HTTPClient	to request a confUri from the ConfTopicMapper given a topic

Figure 4.8. *Publisher and Subscriber function summary*

Server

Server	
ConfFactory	a SIP Conference Server addressed by a FactoryURI
SIPServer	a SIP Server including Registrar and Redirect function
HTTPClient	to update the ConferenceURI-Topic mapping

Figure 4.9. *Server function summary*

The Server in this demo consisted of three components, all operating in the signaling plane. The function summary is listed in Figure 4.9. The ConferenceFactory was a particular SIP server contacted by any Participant in the PubSub system that needed to create a conference for a specific topic. The SIPServer handled the conventional functions of a SIP server, like User Agent registration and session negotiation management. The HTTP-Client was added to update the information regarding topic-conferenceURI mapping in the Mapper using HTTP requests.

Topic Conference Mapper

Topic Conference Mapper	
HTTPServer	to receive update from Server
HTTPClient	to update the IP-addresses at a Media Resource Function
Database	to maintain the Topic <-> Conference mapping

Figure 4.10. *Topic Conference Mapper function summary*

The Topic Conference Mapper(Figure 4.10) is a database equipped with two HTTP modules to command the MRF for RTP-packet forwarding and receive topic-related updates from the Server.

MRF

Media Resource Function-**MRF**	
HTTPServer	to receive update from the Topic Conference Mapper
RTP-MRF	to forward the RTP packets from a Publisher to Subscribers

Figure 4.11. *MRF function summary*

The MRF in this demo was an RTP-Media Resource Function acting as a packet forwarder

with an HTTPClient added for receiving updates from the Topic Conference Mapper. Its function was summarized in Figure 4.11

4.3.2.2 Operation sequences

From the generic operations described in Section 3.2.1, more details were added to formalize the implementable call flows for an IoT PubSub system demo. This part describes all designed call flows, some of which have been implemented. In the following description, the registration procedure was assumed to be set. Therefore, registration call flows were displayed in a short form. Parts of the SDP for SIP messages were placed in the yellow note boxes as a handy convention in this demo. Color coding was used to quickly recognize the relevant information: blue for the topic related, yellow for data format, green for the participant's role, and red for the conference URI.

Topic publishing

Diagram 4.12 illustrates a call flow for the topic publishing operation initiated by a publisher. In flow-3, the publisher sends a SIP INVITE to the Conference Factory function in the Server using the factory's URI. In the attached SDP, the topic was set as `iotTopic` with the text/T140 format. The publisher proposed its role via the SDP attribute `sendonly`. The IP address and port were encoded as in any conventional session negotiation.

A chain of operations (flow-4 to flow-7) using HTTP was made from the Server to the ConfTopicMapper and from the ConfTopicMapper to the MRF. At the end of these operations, the topic-conference URI mapping was updated. In addition, the MRF had the information it needed to receive and forward the RTP packets from the Publisher to the Subscribers.

In flow-8, the Server replied to the Publisher with a SIP 200 OK message. The attached SDP contained the following information:

- The IP address and port of the MRF. For example, notice that the connection attribute has changed from `10.5.0.2` in flow-3 to `10.5.0.3` in flow-8.
- The corresponding conference URI of the conference made for the requested topic. This information was placed in a made-up SDP attribute in this demo.
- The role of the MRF, which was set to `recvonly`

When the session negotiation was done, the Publisher sent RTP packets to the MRF. The topic publishing process was completed.

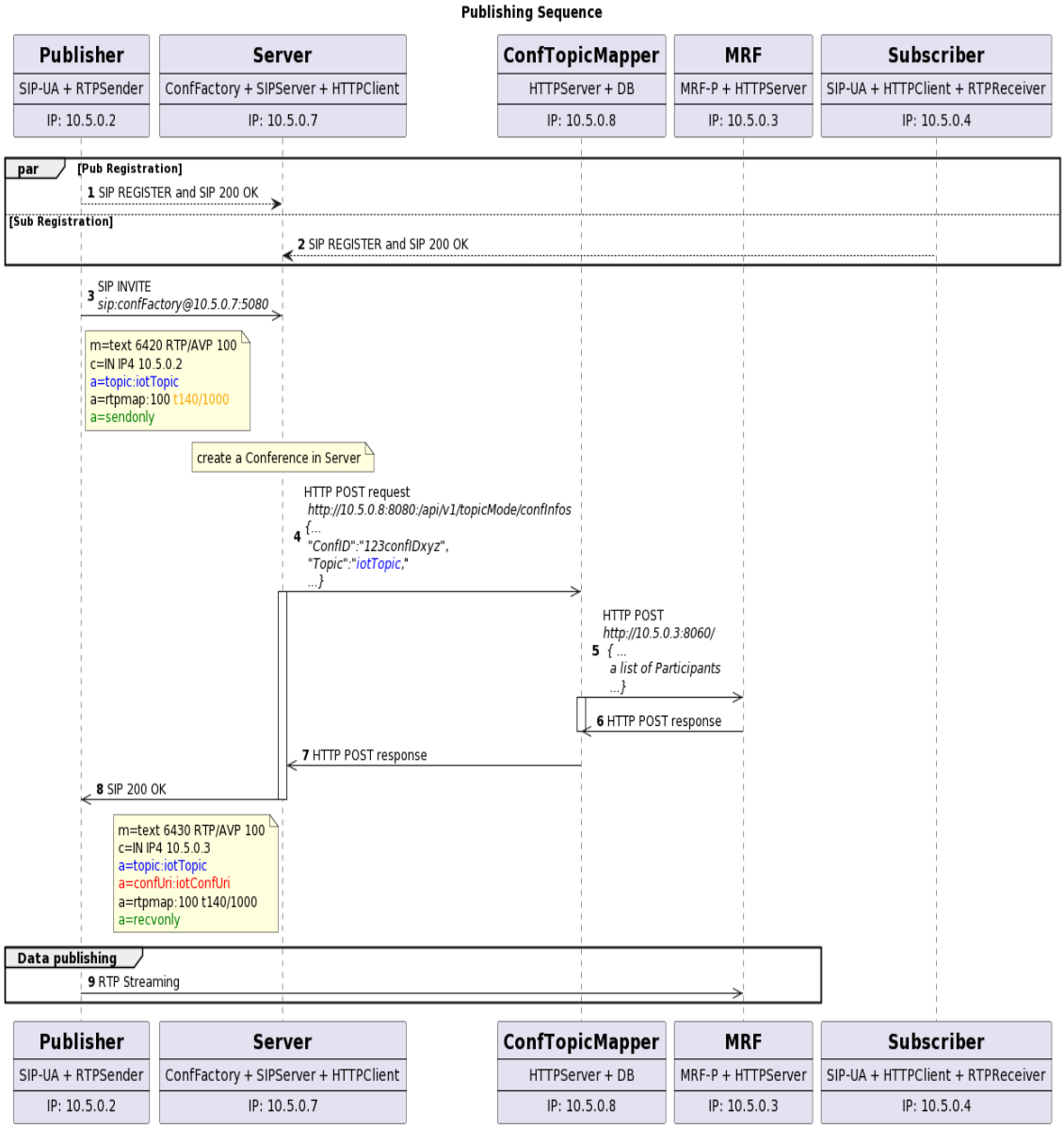

Figure 4.12. *Topic publishing call flow*

Topic subscribing

Diagram 4.13 illustrated the call flow of a subscriber subscribing to a topic. In this call flow, the subscriber was given a topic it was interested in. In flow-1 and flow-2, the subscriber used the topic name to query the conference URI representing that topic. Then, the subscriber joined the found conference by sending a SIP INVITE to the conference URI, similar to how a participant invites himself to an AV conference. It indicated its role as `recvonly` and its IP address:port. In flow-4, the Server replied to the subscriber with a SIP 200 OK message with the IP address-port and the role of the MRF. Flow-5 and flow-6 were for the Server to propagate the forwarding setting to the MRF for the RTP stream forwarding in flow-7. The call flow described the topic subscription for only one subscriber. Still, multiple subscribers can operate the same call flow to that conference and complete

a one-to-many data distributing system.

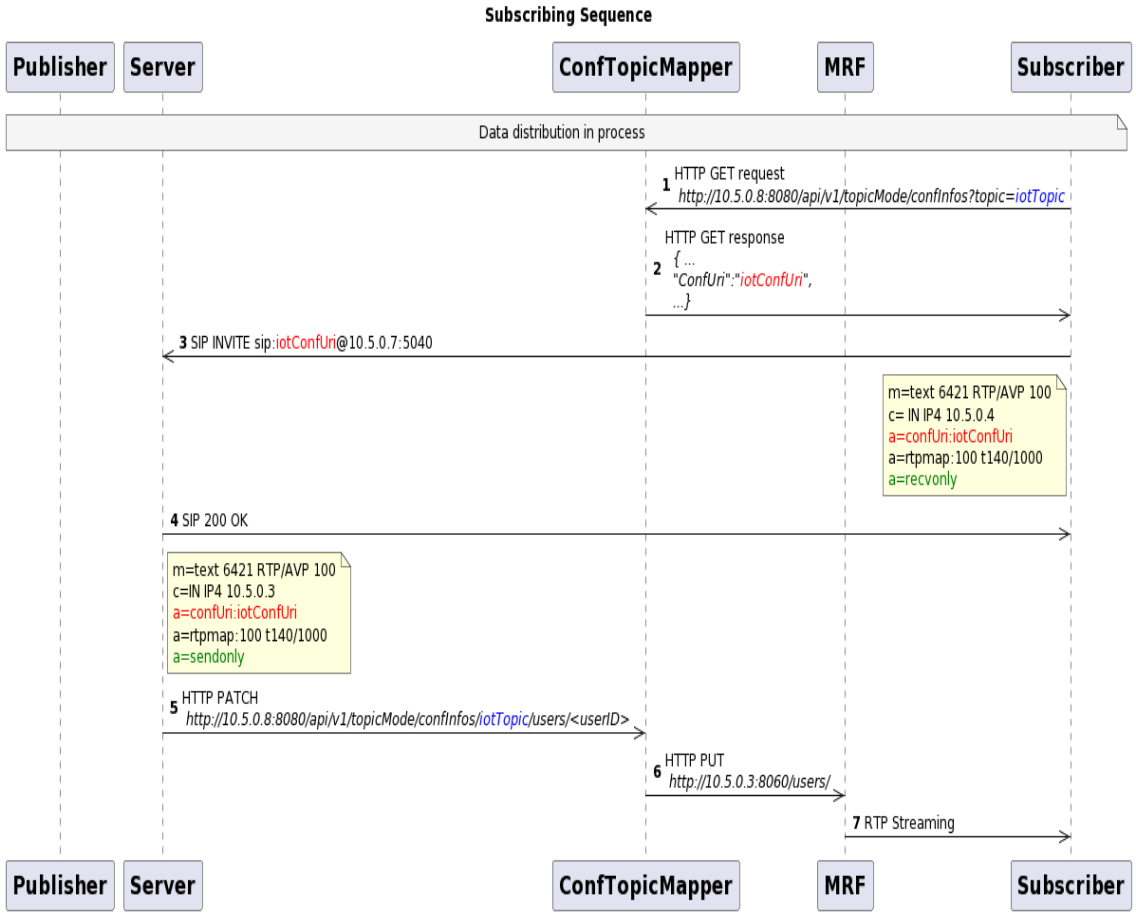

Figure 4.13. *Topic subscribing call flow*

Topic unpublishing

Topic unpublishing is the reversed process of topic publishing in which a publisher deactivates a topic. The call flow a publisher used to unpublish a topic can be seen in Diagram 4.14. The publisher started the unpublishing process by sending a SIP BYE message to the Conference Server function in the Server(flow-1). A sequence of flows from 2 to 5 was made to inform the ConfTopicMapper and MRF about the unpublishing decision. When the Server confirmed the changes were applied to other components, it sent SIP BYE to all subscribers interested in that topic. After that, the server replied to the publisher with a SIP 200 OK(flow-8), and the publisher ended its RTP streaming to the MRF(flow-9).

Depending on specific IoT applications, the call flows can be modified accordingly. For example, if the publisher is limited in battery, it can stop the RTP streaming operation after sending the first SIP BYE in flow 1. This helps the capability-limited publisher save energy while putting extra responsibility in the unpublishing decision.

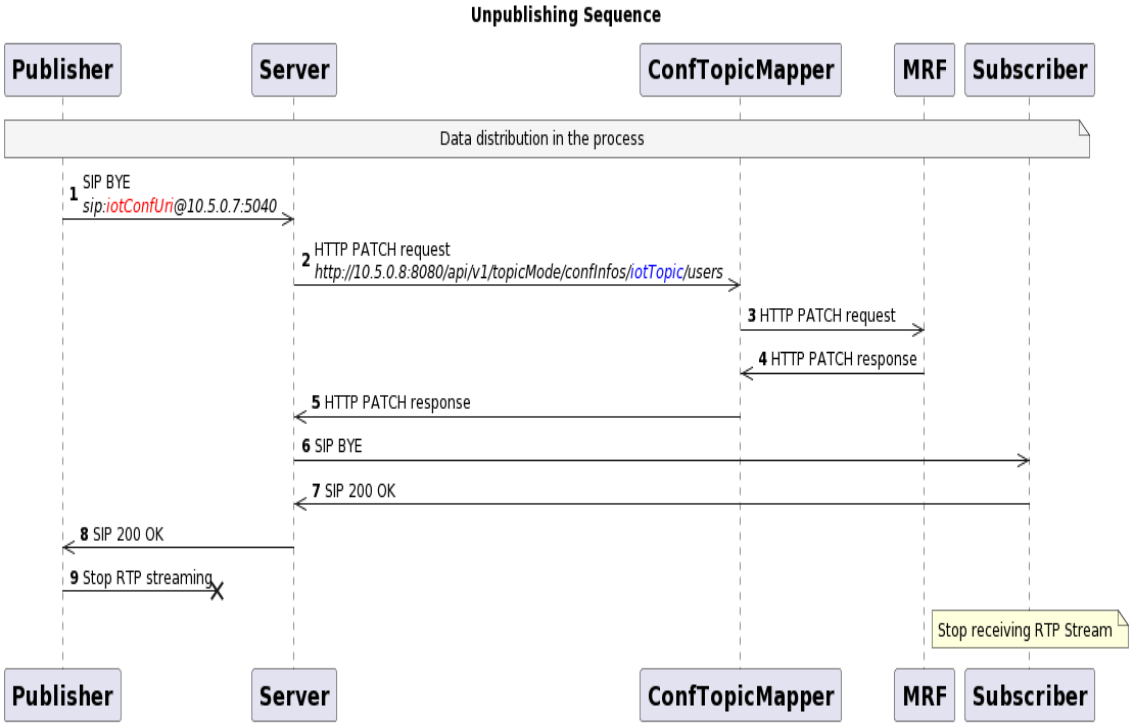

Figure 4.14. *Topic unpublishing call flow*

Topic unsubscribing

The topic unsubscribing procedure was described in Diagram 4.15. The call flow was started by a subscriber sending a SIP BYE to the Conference Server function of the Server(flow-1). After receiving SIP 200 OK from the Server, the subscriber could stop processing RTP packets. There were two scenarios described in Diagram 4.15: the topic could be empty or occupied by one or more subscribers. In the former scenario(flow-4 to flow-9), the Server can send SIP BYE to the publisher to stop publishing data without consumers. In the latter, the publisher and the MRF continued their function.

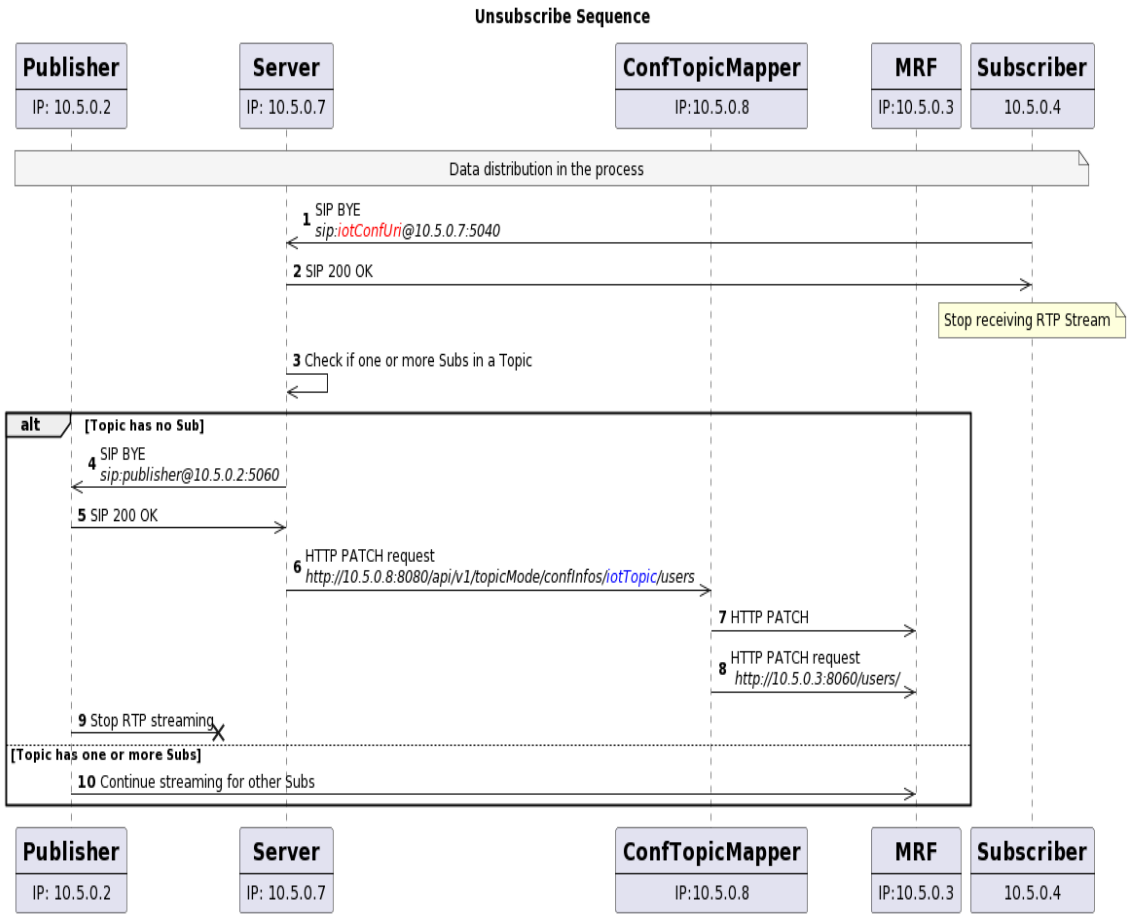

Figure 4.15. *Topic unsubscribing call flow*

4.3.2.3 Architecture

The architecture used in the second demo can be seen in Figure 4.16 and was color coded as follows:

- Orange highlights the components that handle the SIP protocol, including the SIP-Client, SIPServer, and the ConferenceFactory.
- Blue is used for the HTTP implementing functions.
- Green highlights the RTP-related components: Sender, Receivers, and MRF(forwarder).

The repeated details of subscribing process for Subscribers 2 and 3 were left out from Figure 4.16 to improve clarity. It can be seen in the architecture that the Server didn't access the data in the transfer. By initiating MRFs for the transferring tasks, a Server can better react to traffic fluctuation via scaling the MRF number.

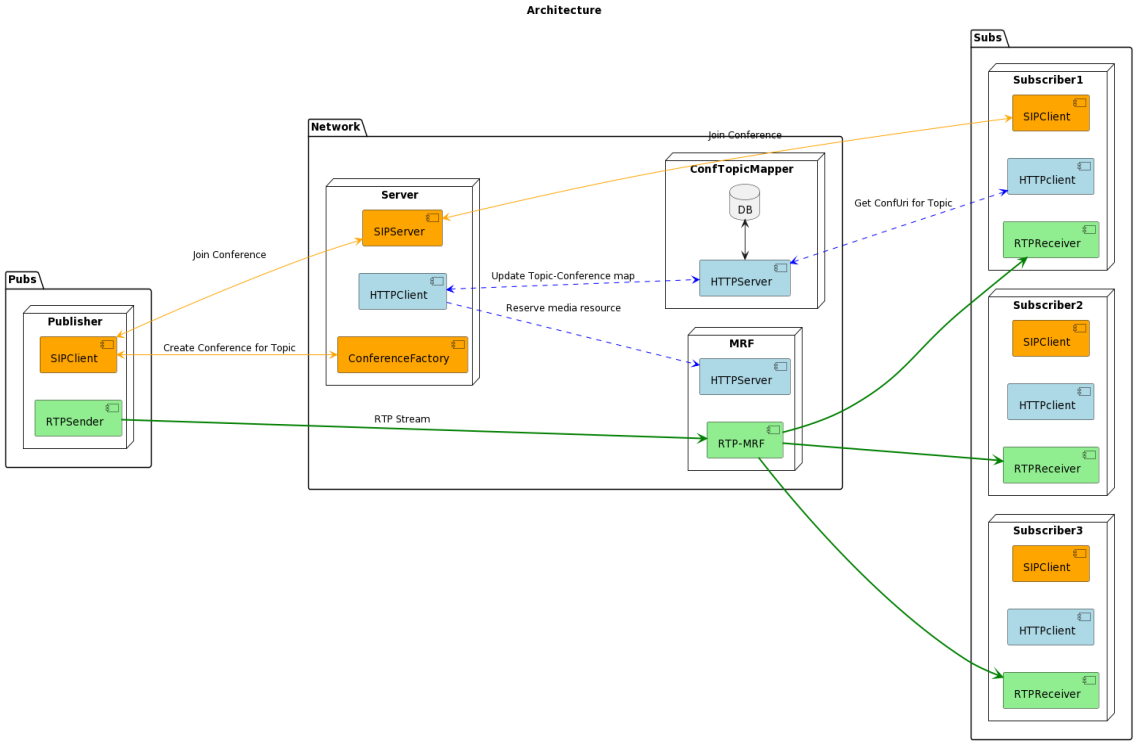

Figure 4.16. *IoT PubSub system architecture*

5. RESULTS AND DISCUSSIONS

5.1 IoT Streaming System

Reflecting on the set targets, the following results have been achieved:

- A system whose devices operate on both signaling and data-transferring planes.
- Reusing an existing RTP payload format to deliver IoT data.
- Streaming IoT data using constraint devices, storing the time-series data, and visualizing in real-time.
- Adding multiplexing and simulcasting functionalities on top of the default streaming.

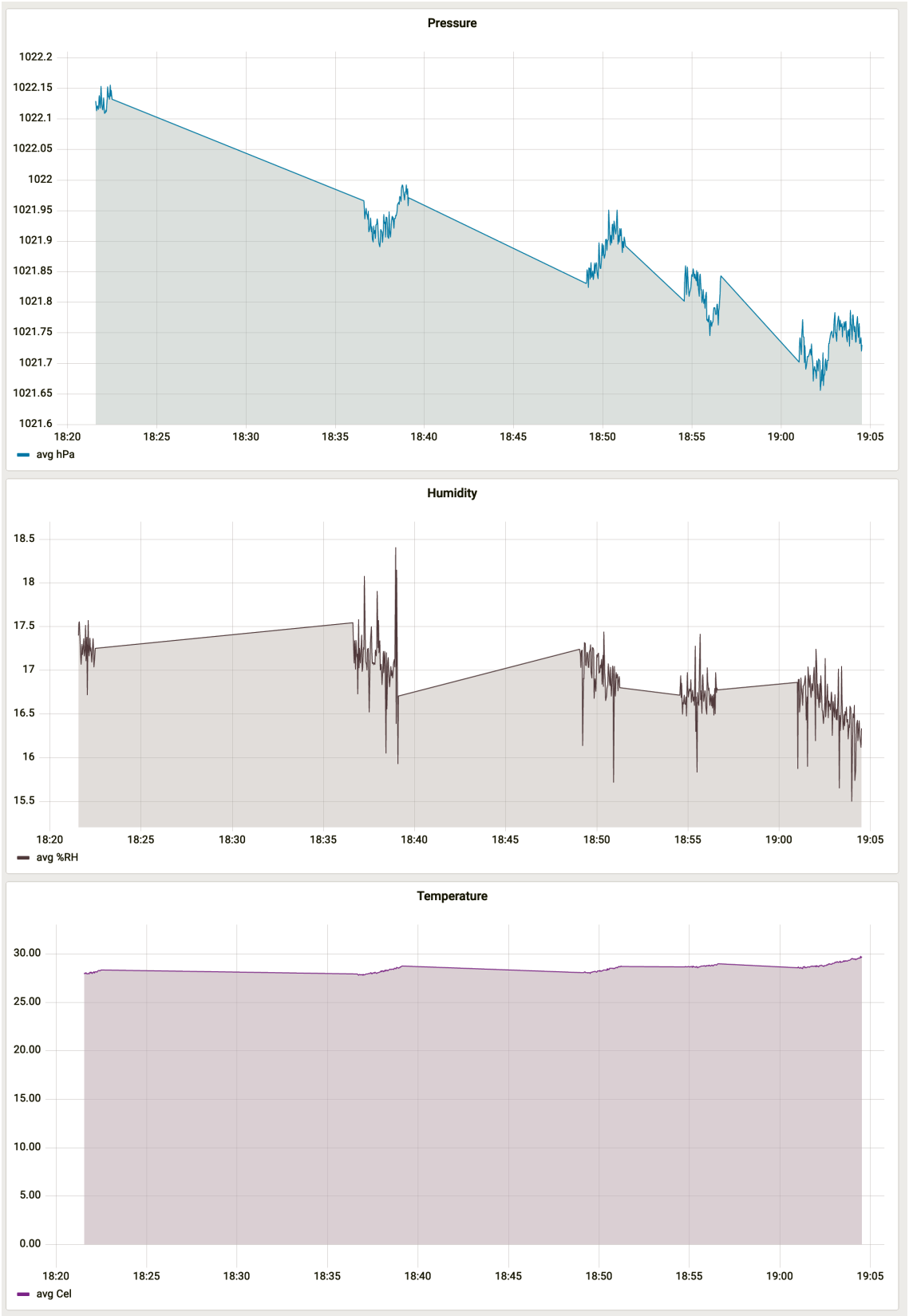

Figure 5.1. *Dashboard capture of IoT streaming demo with multiplex enabled*

Figure 5.1 captured the final result of IoT streaming with the multiplexing function enabled. Three signals were successfully multiplexed at the StreamSender and demultiplexed at

the StreamReceiver. The exact temporal resolution was applied for all three signals. This can be visibly observed in the dashboard for pressure and humidity, where the changes fluctuated in the same periods. Real-time data was stored in the InfluxDB cloud service, and an example of pressure data can be seen in Figure 5.2. The queried data from the InfluxDB aligned with the data observed from the Grafana dashboard. This demo also experimented with a stable end-to-end setting for future prototypes regarding streaming IoT data using RTP.

Figure 5.2. *Pressure data queried from an InfluxDB instance*

For the simulcast function, the final result dashboard for temperature measurement in different resolutions can be seen in Figure 5.3. The interference in hardware was assumed to cause outliers at around 300 degrees and zero degrees in both dashboards. However, no packet loss was detected, and signals in both resolutions were successfully delivered to the visualizing components.

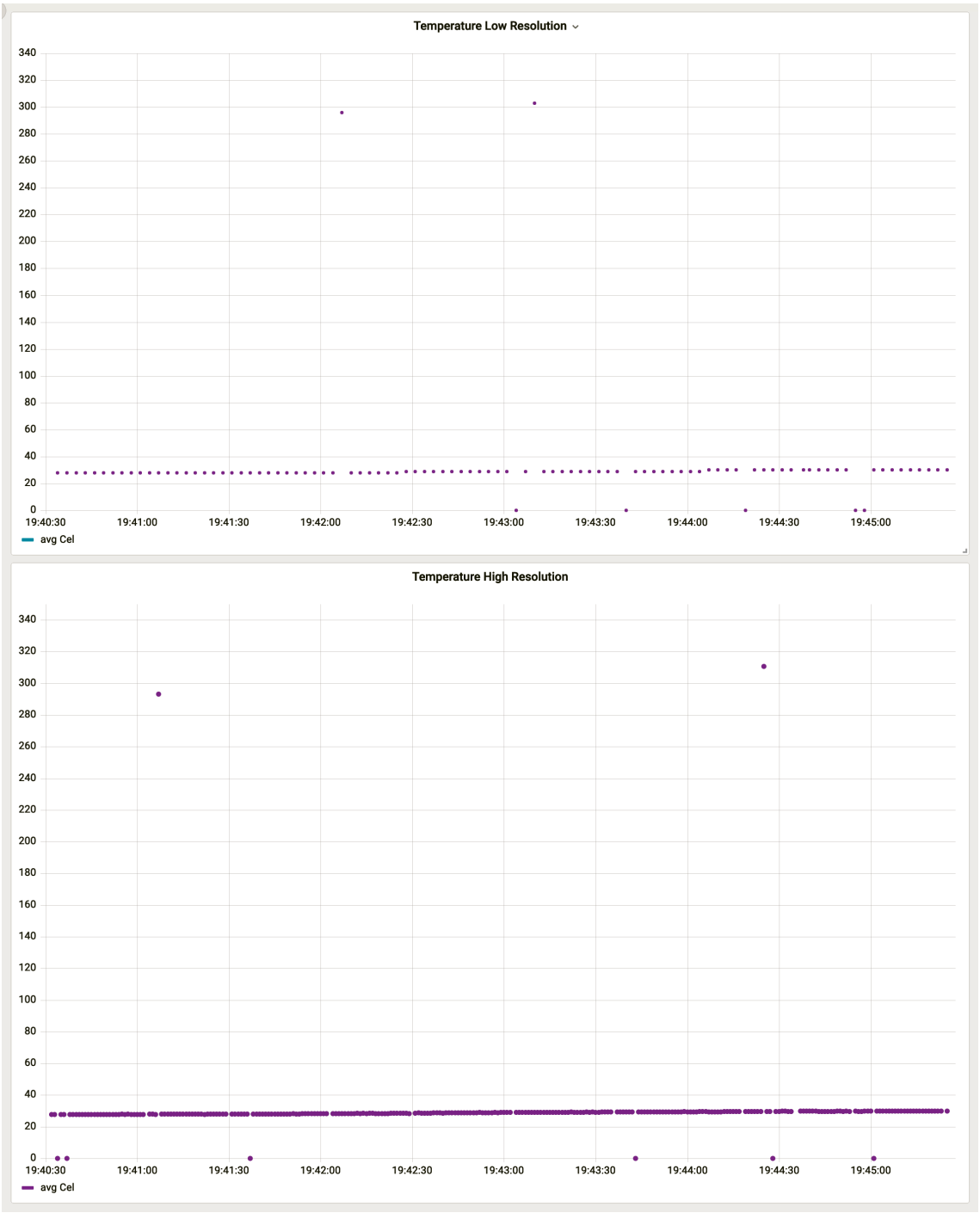

Figure 5.3. *Dashboard capture of IoT streaming demo with simulcast enabled*

The signal in the high-resolution dashboard was updated more frequently. In addition, by zooming in to a shorter period, as in Figure 5.4, the higher precision can also be seen from the subtle changes in the second dashboard compared with the constant value in the first one.

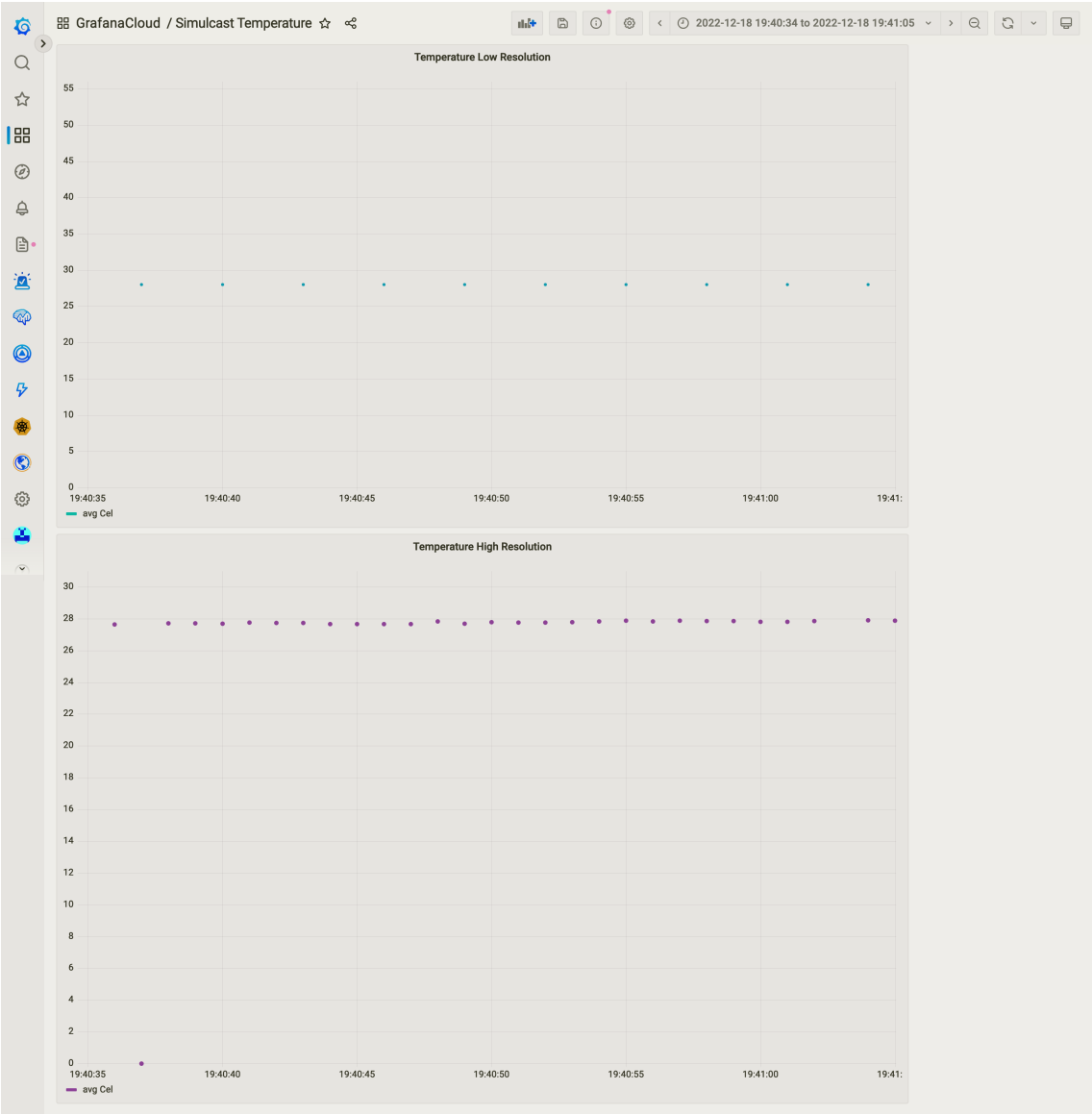

Figure 5.4. *Closer look into the simulcasting dashboard*

From these results, it can be seen that streaming IoT data over RTP does not require major changes in RTP or SIP. Given an AV streaming system, most of the work to expand to IoT streaming lay on the IoT packetizer and depacketizer. Various data-publishing frequencies have been set to examine the prototype's capability for different data temporal requirements. The observation showed that the data rate could be adjusted flexibly once the session modification was handled. A stream sender can publish data concurrently while waiting to negotiate a new valid stream setting.

The SSRC-multiplexing from the AV streaming can be reused directly for IoT streams. To enable the simulcast function, an additional step was to define the "resolution" for IoT streams, as discussed in Section 4.

5.2 IoT Pubsub System

The final result of the IoT PubSub demo satisfied the following bullet points:

- Reusing the SIP session negotiation and RTP streaming for the IoT PubSub implementation as mentioned in `Target 1`
- Enabling multiple PubSub operations listed in `Target 2`
- Implementing the Topic-Conference URI Mapper with HTTP interface.

The topic unsubscribing procedure didn't strictly follow the designed call flow mentioned. Instead of sending a SIP BYE to the Server, the subscriber sent an HTTP PATCH request to the ConfTopicMapper to mimic the SIP call flow. Future work can be allocated to complete the designed call flows.

5.3 Other insights

The prototype implementation of IoT Streaming and IoT PubSub using RTP, SIP, and PubSub technologies yielded several shared insights. First, under test conditions without packet loss, RTP demonstrated its capability to deliver IoT data with low latency while preserving all stream characteristics, making it a promising technology for real-time IoT data transmission. However, using SIP for signaling tasks may be too demanding for some IoT devices, and many of its features may remain unused in IoT data streaming sessions. The implementation also encountered practical issues, such as signal smoothing and outlier handling, which were not initially anticipated, underscoring the importance of thoroughly testing and validating IoT streaming systems. Additionally, compared to other fields such as web or machine learning, the number of IoT-focused projects is relatively small, and unstable hardware can impede prototype development, indicating a need for improved hardware stability and IoT-oriented platforms that can support data generation, aggregation, and system integration. In summary, these insights highlight the importance of addressing practical implementation issues when developing IoT streaming systems and underscore the need for further research and development in this area.

6. CONCLUSIONS

Based on the research in this thesis work, the following claims were stated:

`Question 1` How can the RTP built-in features and extensions be utilized for IoT Streaming?

The RTP/SIP stack available in AV streaming has numerous built-in features and extensions that can be effectively applied to IoT streaming. By leveraging these features, many IoT use cases can be realized with low latency and high reliability. The flexibility and scalability of RTP make it an ideal candidate for IoT data delivery, while SIP provides a reliable signaling plane for establishing and managing IoT data streaming sessions. Furthermore, the existing RTP extensions can be customized to support specific IoT use cases. This approach provides a powerful framework for implementing real-time IoT applications with high performance and low overhead.

`Question 2` How the A/V infrastructure can be reused for IoT Streaming?

Implementing IoT Streaming over RTP in the VoLTE/IMS infrastructures can bring a multitude of benefits for all the involved parties. Firstly, end-users can benefit from RTP's low-latency and high-quality data delivery, enabling seamless and real-time interaction with IoT devices. Secondly, network service providers can expand their service offerings to include IoT-specific services, catering to the growing demand in the market. This can lead to new revenue streams and increased customer loyalty. Additionally, the utilization of existing VoLTE/IMS infrastructure can reduce deployment costs and time-to-market for new IoT services. Lastly, this implementation can pave the way for future advancements and innovations in the IoT domain by providing a reliable and scalable solution for IoT data delivery over telecommunication networks.

`Question 3` How the A/V Conference Services can be used to realize an IoT Pub-Sub Service?

Implementing each Conference as a Topic in the PubSub architecture with SIP for signaling and RTP for data transferring functions is a sig-

nificant step towards enabling effective data distribution and interaction among IoT devices. This approach allows IoT devices to seamlessly join and participate in an IoT Conference, benefiting from PubSub behaviors for efficient data distribution while still enjoying RTP's low latency and stream-supported data delivery. This implementation is a testament to the feasibility and effectiveness of the proposed approach and can serve as a blueprint for future developments in IoT data distribution and interaction.

`Addition`

The minimal changes required for implementing IoT Streaming and IoT PubSub over RTP and SIP/SDP are promising for telecommunication service providers and communication platform providers. These streamlined implementations provide a solid foundation for the rapid rollout of IoT applications. By leveraging the existing infrastructure and protocols, these providers can offer new IoT services to their customers quickly and cost-effectively. The ease of implementation also allows for custom IoT solutions tailored to specific customer needs. This opens new revenue streams for service providers and enhances customer satisfaction by providing them with personalized and innovative solutions. Overall, the minimal implementation requirements for IoT Streaming and IoT PubSub underscore the potential of these technologies for revolutionizing the IoT landscape.

During the IoT realization and prototyping phase, several promising ideas have emerged for future work on IoT streaming. These ideas can be broadly classified into two groups: experimental and standardization. On the experimental front, further prototyping and experimentation on real-world VoLTE networks are needed to more thoroughly assess the reusability of the current audio-video infrastructure for IoT streaming. Another direction would be to explore the use of other payload types for RTP packets, such as robot commands or OPC UA payloads, or investigate AV and IoT data multiplexing to support XR-related applications.

On the standardization front, ongoing work is critical for bringing IoT streaming applications to the public. For example, developing a new SDP attribute for a topic and a new SIP event package specified for IoT PubSub implementation in Chapter 4 would be essential steps toward standardizing IoT streaming capabilities. Future work in VoLTE standardization for IoT streaming is also necessary to ensure seamless integration of these new IoT capabilities with prior standards.

In summary, these future directions underscore the need for continued research and development in IoT streaming to realize its potential fully. The experimental work will enable

the refinement and optimization of existing technologies for IoT streaming, while standardization efforts will promote broader adoption and interoperability across different systems and networks. We can advance state-of-the-art IoT streaming by pursuing these avenues and help unlock new possibilities for various applications and use cases.

A. TECHNICAL DETAILS

A.1 RTP redundancy functionality

To assure the association of the redundant streams and their original stream, if the RTP implementation is using SDP for session negotiation, then either one of the two following conditions must be met:

- If there exist other streams other than the original and its duplicate, the description of the original and its duplicate (with different SSRCs) must be put on the same "m" line.
- The original and its duplicate can only be described using different "m" lines if they are the only streams described on each of those lines.

Considering the feedback perspective, RTCP packets for the duplicate must be generated without discrimination with the original. Reusing the reports from the original for the duplicate is forbidden as those don't reflect the true condition of the session and almost certainly lead to undesired behaviors. On the signaling plane, there are two requirements to enable RTP redundancy:

- Participants need to include the redundancy supporting their negotiation for session setting (via SDP extension defined in the RFC7197[85]).
- The duplicate must be specified via signaling means either in-band via RTCP or out-of-band using SDP.

A.2 RTP pause/resume functionality

The basic usages of pause-resume can be described as:

Request to Pause An RTP receiver can send a PAUSE request at any moment allowed by the time rule in [38]. If a PAUSE request is acknowledged as lost, a new PAUSE request with the same PauseID will be sent.

Request to Resume An RTP receiver can send a RESUME request at any time with the same timing rule applied for PAUSE. The current PauseID is mandatory for any actionable RESUME request.

RTP receiver can send a PAUSE request to an RTP sender at any time because of a requested pause or its internal decision. The ongoing tasks and necessary preparation can be finished before processing the approved pause operation. A PAUSED indication is sent, including the current PauseID and the highest sequence number of the RTP. The latter information is beneficial to pre-pausing packet loss. The paused condition is also signaled to all attending and newly joint participants.

Response on Paused If the targeted RTP sender is already paused with the same current PauseID as the received request, that request will be ignored. Otherwise, the sending continues for a short time (called a hold-off period) while waiting for other senders' consensus on the PAUSE request. If there is a disapproving RESUME during the hold-off period, no pausing on the receiver is to be executed, but the PauseID is still updated. If the PauseID in the PAUSE request does not match the current PauseID, a REFUSED notification with the current PauseID value will be sent. REFUSED notification can signal an RTP sender's local decision not to pause, indicated by the same PauseID in the received request.

Response on Resume A being paused RTP sender resumes its stream if the PauseID in the RESUME request is the current value. An RTP sender with an active stream ignores the received RESUME. Similar to the case of the PAUSE request, the local logic also contributes to the sender's final decision on a RESUME request.

A.3 RTP Inter-Destination Multimedia Sync

Regarding inter-destination sync, in RFC7272 [86] R. van Branderburg proposed an architecture for IDMS (Inter-Destination Media Synchronization).

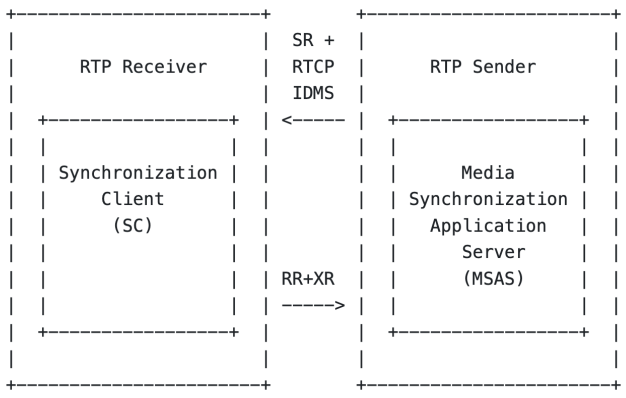

Figure A.1. *IDMS Architecture Diagram. Adopted from [86]*

As can be seen from Figure A.1, there are two new entity types named Synchronization Client (SC) and Media Synchronization Application Server (MSAS). An SC is placed inside each RTP receiver and responsible for: reporting packet arrival time, packet repre-

senting time, and configuring playout function based on synchronization setting from the MSAS. The sync setting contains synchronizing instructions for the RTP receivers. An MSAS can be realized as an additional function at the RTP sender or an independent function in an RTP implementation to collect the sync-related reports from SCs. Based on these reports, MSAS creates and distributes the sync setting to its SCs. To provide an official means to convey feedback information on inter-destination sync, RFC7272 also defined:

- An RTCP XR block for IDMS to use in the SC $\rightarrow$ MSAS communication (Figure A.3).
- An RTCP Packet Type for indicating the sync setting for RTP receivers in the MSAS $\rightarrow$ SC communication (Figure A.2).

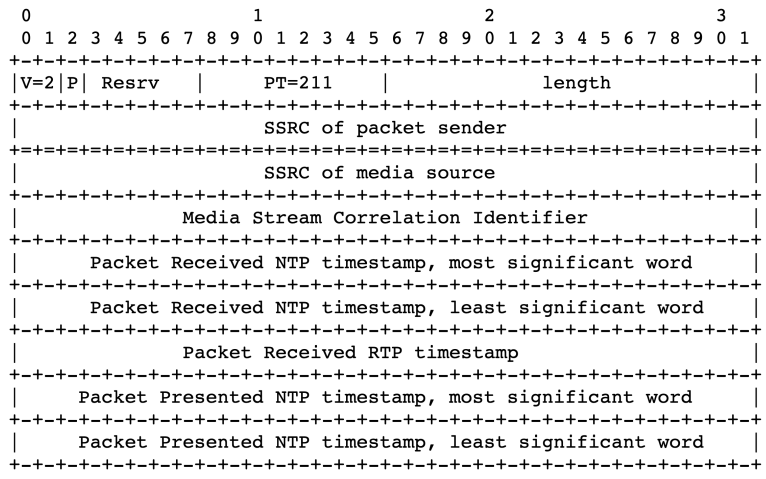

Figure A.2. *RTCP IDMS setting packet format. Adopted from [86]*

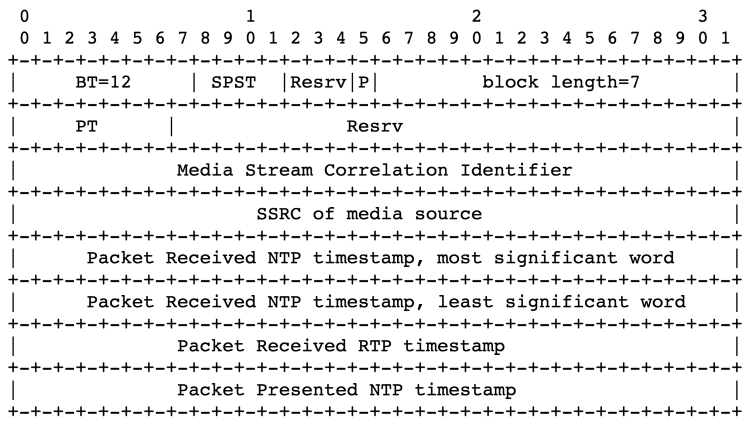

Figure A.3. *RTCP IDMS XR report format. Adopted from [86]*

A note should be made that the most constrained sync setting is chosen for the RTP receivers to ensure all receivers are in sync. As can be seen from the work of Fernando

Boronat Segui in 2008 [87] and Mario Montagud in 2012 [88], sufficient IDMS function can already be implemented using only their user-defined RTCP extension (such as in the RTCP APP and RR reports). However, the usage of standardized solutions like RFC7272 was proved to be more effective in [89] with the improvement as:

- More flexible selection for the wall clock source.
- Support for both packet reception time and presentation time.
- A high resolution (64 bits) for `Packet Received NTP timestamp` and `Packet Presented NTP timestamp` in RTCP IDMS setting packet to set up the synchronization.
- A larger namespace (32 bits) for group ID (`Media Stream Correlation Identifier`).
- A higher accuracy for RTCP-to-RTP mapping using RTP timestamps instead of RTP sequence numbers.

For the advantages listed above, the approach using RFC7272 should be prioritized for an IDMS solution.

A.4 VoLTE

Feature	VoLTE requirements	3GPP requirements
Supported UE IP versions	Both IPv4 and IPv6	Both IPv4 and IPv6 IPv4 alone IPv6 alone
Supported UE voice domain preferences	IMS only IMS preferred/CS secondary	IMS only IMS preferred/CS secondary CS preferred/IMS secondary CS only
Support of MMTel voice service	Mandatory	Optional
Support of AMR voice codec	Mandatory	Mandatory if MMTel voice is supported
Support of MMTel supplementary services	Some mandatory services	All services optional
Support of emergency calls	Mandatory	Optional
Support of SMS	Mandatory	Optional
Support of SRVCC	Mandatory if CS is supported	Optional
IMS access point name	IMS well-known APN	Any APN
Location of PGW and PCSCF	Visited network	Home or visited network
PCSCF discovery technique	During default bearer activation	Four techniques supported
Transport of SIP signalling	Default bearer (QCI 5)	Any bearer and QCI
Transport of VoIP traffic	One dedicated bearer (QCI 1)	Any bearer and QCI
VoIP SDF establishment	Triggered by PCSCF	Triggered by PCSCF or UE

Figure A.4. *Key differences in VoLTE specifications between 3GPP and GSMA. Adopted from [90]*

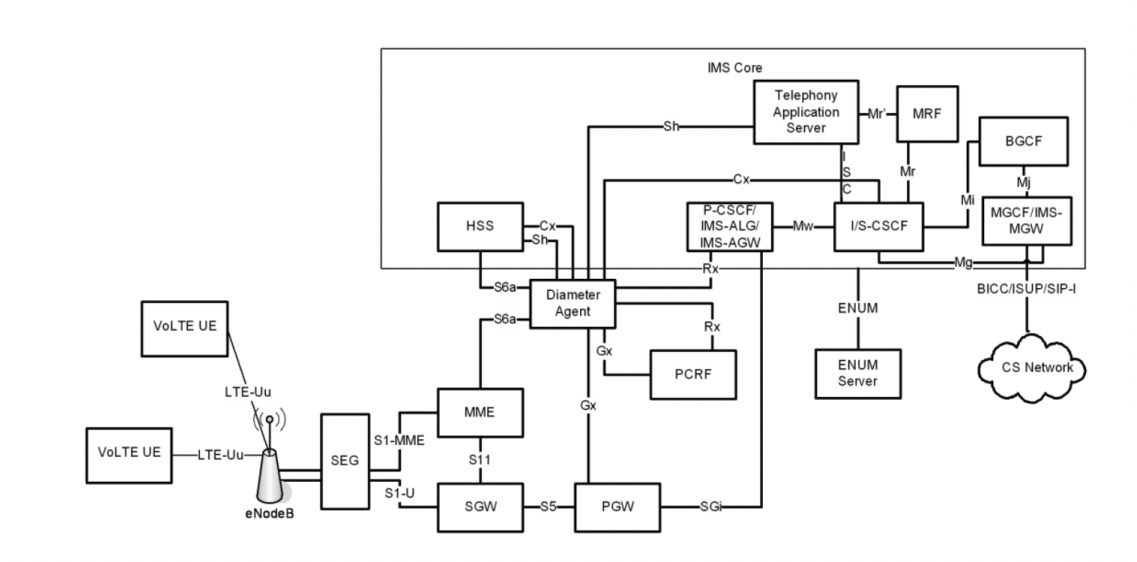

Figure A.5. *Intra-PMN VoLTE deployment. Adapted from[51]*

A.5 Conference

This appendix includes a description of other types of conference architecture specified in RFC4353.

Endpoint Server

In the Endpoint Server model, a conference has two essential properties: ad-hoc and locally-mixed. At the start, a regular point-to-point call is set up between two users. During the call, there may occur demands for conference communication like: more Participants want to join, a Participant intends to record the session or there are files to be shared. A regular call then goes through the Transition process (the arrow in Figure A.6) to be replaced by a conference. One of the users in that call implements the Focus function in addition to its single UA function (becoming an Endpoint Server). That Endpoint Sever obtains a new URI and uses that in the INVITE sent to additional users. The scenario that a conference was not planned and started from a call makes the ad-hoc property. Figure A.6 suggests that the Mixers are located only within one Participant. Therefore, all media streams from other Participants are mixed within the Endpoint Server and then re-distributed.

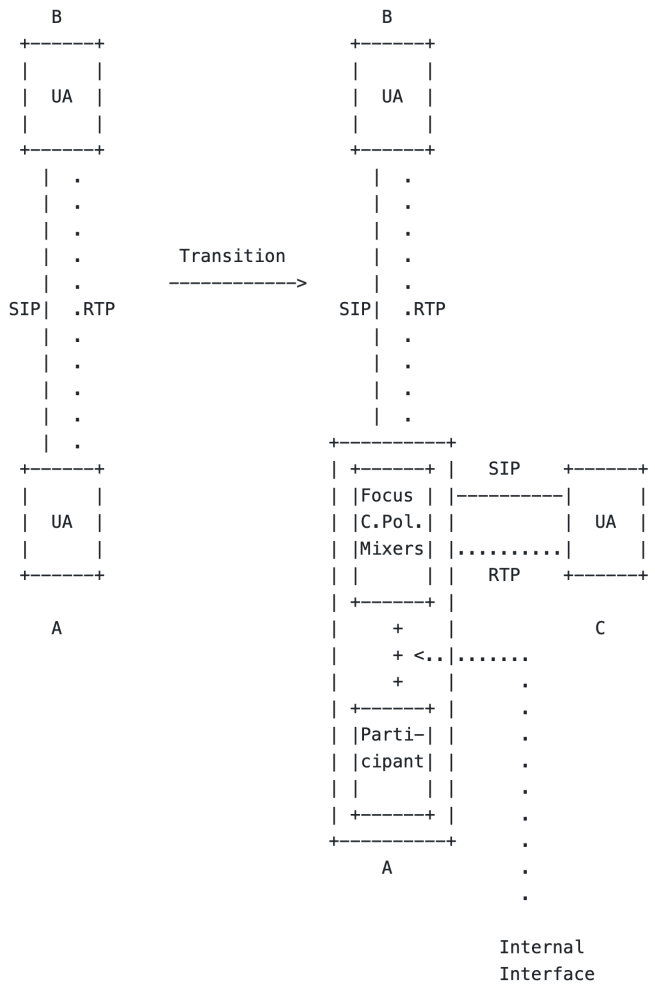

Figure A.6. *Endpoint Server Model. Adopted from [55]*

Media Server Component

In this model, server functions, as presented in Endpoint Server, are divided into two component servers:

> Application Server for interaction with Participants. The Conference Notification Service is exclusive to the Application Server.
>
> Mixing Server for media mixing tasks. Mixers are only available on a Mixing Server.

Each Server has its Focus and Conference Policy but uses them slightly differently. In an Application Server, the Focus interacts with other Participants on the Signaling Plane and updates its Conference Policy accordingly. A Mixing Server uses its Focus solely to receive control commands from its Application Server. These commands connect media streams to Participants, adjust the mixing process, and update the Conference Policy. This separation allows reusing a single Mixing Server for different Application Servers.

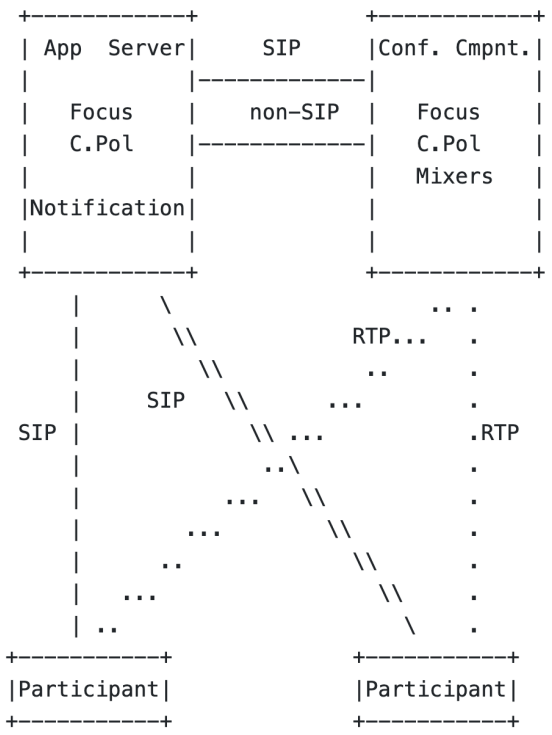

Figure A.7. *Media Server Model. Adopted from [55]*

Distributed Mixing

A Conference Server in this model only handles the signaling functions and distributes all media functions to conference Participants. As can be seen from Figure A.8, each Participant has its Mixers and Conference Policy Server. This grants an individual Participant a more flexible media mixing and policy negotiation with the Focus. On the other hand, additional functionality in each UA is also required.

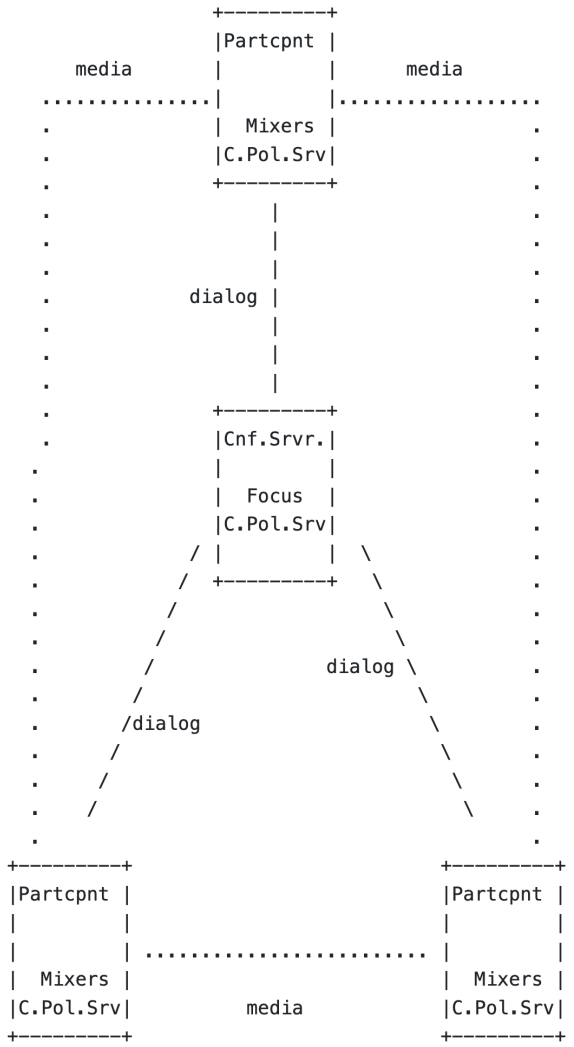

Figure A.8. *Distributed Mixing Model. Adopted from [55]*

Cascaded Mixer

In the Cascaded Mixers model, operations on the Signal Plane are organized similarly to the Centralized Server and done by the SIP mechanism. The media mixing function on the Data Transfer Plane is scattered into different Mixers throughout the Conference network. An individual Participant is assigned to only one Mixer. All Mixers are under Focus' orchestration to perform the correct mixing based on the Conference Policy stored in the Conference Server. The control protocol is left unspecified in RFC4353.

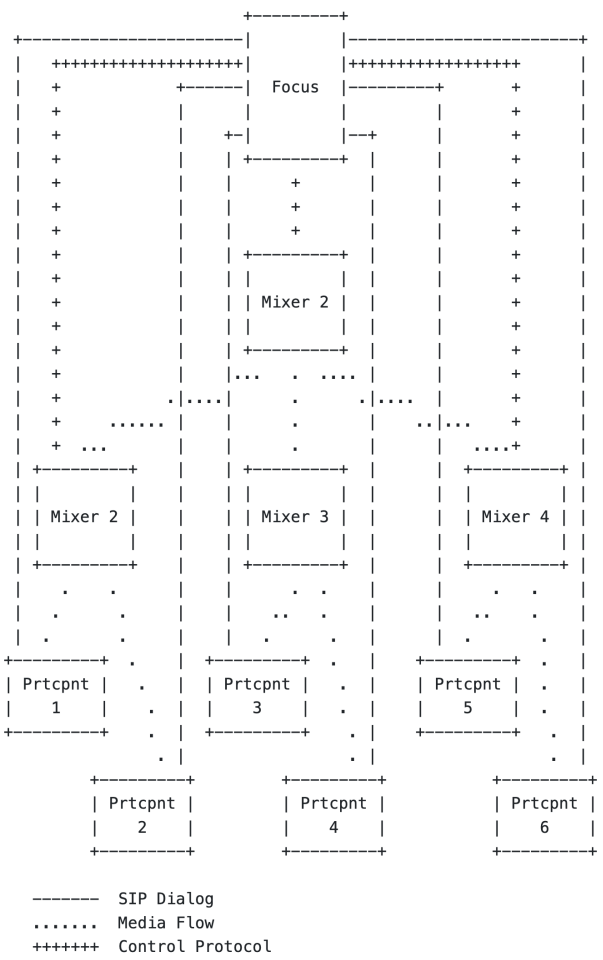

Figure A.9. *Cascaded Mixer Model. Adopted from [55]*

REFERENCES

[1] Trayush, T., Bathla, R., Saini, S. and Shukla, V. K. IoT in Healthcare: Challenges, Benefits, Applications, and Opportunities. *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. 2021, pp. 107–111. DOI: 10.1109/ICACITE51222.2021.9404583.

[2] Ayaz, M., Ammad-Uddin, M., Sharif, Z., Mansour, A. and Aggoune, E.-H. M. Internet-of-Things (IoT)-Based Smart Agriculture: Toward Making the Fields Talk. *IEEE Access* 7 (2019), pp. 129551–129583. DOI: 10.1109/access.2019.2932609. URL: https://doi.org/10.1109/access.2019.2932609.

[3] Ray, P. P. Internet of Robotic Things: Concept, Technologies, and Challenges. *IEEE Access* 4 (2016), pp. 9489–9500. DOI: 10.1109/access.2017.2647747. URL: https://doi.org/10.1109/access.2017.2647747.

[4] Cisco. *Cisco Annual Internet Report (2018-2023)*. 2020. URL: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf.

[5] Cisco. *VNI Complete Forecast Highlights*. 2018. URL: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_Device_Growth_Traffic_Profiles.pdf.

[6] Alvi, S. A., Afzal, B., Shah, G. A., Atzori, L. and Mahmood, W. Internet of multimedia things: Vision and challenges. *Ad Hoc Networks* 33 (Oct. 2015), pp. 87–111. DOI: 10.1016/j.adhoc.2015.04.006. URL: https://doi.org/10.1016/j.adhoc.2015.04.006.

[7] Turchet, L., Fazekas, G., Lagrange, M., Ghadikolaei, H. S. and Fischione, C. The Internet of Audio Things: State of the Art, Vision, and Challenges. *IEEE Internet of Things Journal* 7.10 (2020), pp. 10233–10249. DOI: 10.1109/JIOT.2020.2997047.

[8] Turchet, L., Fischione, C., Essl, G., Keller, D. and Barthet, M. Internet of Musical Things: Vision and Challenges. *IEEE Access* 6 (2018), pp. 61994–62017. DOI: 10.1109/ACCESS.2018.2872625.

[9] Karaagac, A., Camerlynck, P., Crombez, P. and Hoebeke, J. VIoT: Voice over Internet of Things. *2020 Global Internet of Things Summit (GIoTS)*. IEEE, June 2020. DOI: 10.1109/giots49054.2020.9119616. URL: https://doi.org/10.1109/giots49054.2020.9119616.

[10] Lin, J. The lambda and the kappa. *IEEE Internet Computing* 21.05 (2017), pp. 60–66.

[11] Karakaya, Z., Yazici, A. and Alayyoub, M. A Comparison of Stream Processing Frameworks. *2017 International Conference on Computer and Applications (ICCA)*. 2017, pp. 1–12. DOI: 10.1109/COMAPP.2017.8079733.

[12] Vijaykumar, S. and T, S. P. A Literature Survey on Various Streaming Protocol and Compression Techniques in Multimedia Transmission with Internet of Things. *2021 International Conference on Computer Communication and Informatics (ICCCI)*. 2021, pp. 1–6. DOI: 10.1109/ICCCI50826.2021.9457018.

[13] Said, O., Albagory, Y., Nofal, M. and Al Raddady, F. IoT-RTP and IoT-RTCP: Adaptive Protocols for Multimedia Transmission over Internet of Things Environments. *IEEE Access* 5 (2017), pp. 16757–16773. DOI: 10.1109/ACCESS.2017.2726902.

[14] Bansode, S. S., Mahajan, R. and Vyas, V. Performance Analysis and Implementation of Adaptive Protocols of IOT Environment. *2021 6th International Conference for Convergence in Technology (I2CT)*. 2021, pp. 1–4. DOI: 10.1109/I2CT51068.2021.9418072.

[15] Herrero, R. MQTT-SN, CoAP, and RTP in wireless IoT real-time communications. *Multimedia Systems* 26.6 (July 2020), pp. 643–654. DOI: 10.1007/s00530-020-00674-5. URL: https://doi.org/10.1007/s00530-020-00674-5.

[16] Cirani, S., Picone, M. and Veltri, L. A session initiation protocol for the Internet of Things. *Scalable Computing: Practice and Experience* 14.4 (Jan. 2014). DOI: 10.12694/scpe.v14i4.931. URL: https://doi.org/10.12694/scpe.v14i4.931.

[17] Cirani, S. and Veltri, L. Lightweight Session Initiation for the Internet of Things. *2015 IEEE Global Communications Conference (GLOBECOM)*. 2015, pp. 1–6. DOI: 10.1109/GLOCOM.2015.7417242.

[18] *Recommendation ITU-T Y4000/Y2060*. 2012.

[19] Ghinea, G. and Thomas, J. Quality of perception: user quality of service in multimedia presentations. *IEEE Transactions on Multimedia* 7.4 (2005), pp. 786–789. DOI: 10.1109/TMM.2005.850960.

[20] Akhtar, Z. and Falk, T. H. Audio-Visual Multimedia Quality Assessment: A Comprehensive Survey. *IEEE Access* 5 (2017), pp. 21090–21117. DOI: 10.1109/ACCESS.2017.2750918.

[21] ITU. *Technical Specification D4.4 Framework to support data quality management in IoT*. July 2019. URL: https://www.itu.int/dms_pub/itu-t/opb/fg/T-FG-DPM-2019-4.4-PDF-E.pdf.

[22] Zhang, L., Jeong, D. and Lee, S. Data Quality Management in the Internet of Things. *Sensors* 21.17 (Aug. 2021), p. 5834. DOI: 10.3390/s21175834. URL: https://doi.org/10.3390/s21175834.

[23] *Who cares about latency in 5G?* URL: https://www.ericsson.com/en/blog/2022/8/who-cares-about-latency-in-5g (visited on 09/30/2022).

[24] Schulzrinne, H., Casner, S. L., Frederick, R. and Jacobson, V. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. July 2003. DOI: 10.17487/RFC3550. URL: https://www.rfc-editor.org/info/rfc3550.

[25] *Ccnp collaboration claccm 300-815 cert guide*. Cisco Press, 2020. ISBN: 0-13-657554-4.

[26] Perkins, C. and Westerlund, M. *Multiplexing RTP Data and Control Packets on a Single Port*. RFC 5761. Apr. 2010. DOI: 10.17487/RFC5761. URL: https://www.rfc-editor.org/info/rfc5761.

[27] Clark, D. D. and Tennenhouse, D. L. Architectural Considerations for a New Generation of Protocols. SIGCOMM '90 (1990), pp. 200–208. DOI: 10.1145/99508.99553. URL: https://doi.org/10.1145/99508.99553.

[28] Deering, D. S. E. *Host extensions for IP multicasting*. RFC 1112. Aug. 1989. DOI: 10.17487/RFC1112. URL: https://www.rfc-editor.org/info/rfc1112.

[29] Schooler, E., Ott, J. and Chesterfield, J. *RTP Control Protocol (RTCP) Extensions for Single-Source Multicast Sessions with Unicast Feedback*. RFC 5760. Feb. 2010. DOI: 10.17487/RFC5760. URL: https://www.rfc-editor.org/info/rfc5760.

[30] Westerlund, M., Burman, B., Perkins, C., Alvestrand, H. T. and Even, R. *Guidelines for Using the Multiplexing Features of RTP to Support Multiple Media Streams*. RFC 8872. Jan. 2021. DOI: 10.17487/RFC8872. URL: https://www.rfc-editor.org/info/rfc8872.

[31] Begen, A. C. and Perkins, C. *Duplicating RTP Streams*. RFC 7198. Apr. 2014. DOI: 10.17487/RFC7198. URL: https://www.rfc-editor.org/info/rfc7198.

[32] Begen, A. C., Cai, Y. and Ou, H. *Duplication Grouping Semantics in the Session Description Protocol*. RFC 7104. Jan. 2014. DOI: 10.17487/RFC7104. URL: https://www.rfc-editor.org/info/rfc7104.

[33] Burman, B., Westerlund, M., Nandakumar, S. and Zanaty, M. *Using Simulcast in Session Description Protocol (SDP) and RTP Sessions*. RFC 8853. Jan. 2021. DOI: 10.17487/RFC8853. URL: https://www.rfc-editor.org/info/rfc8853.

[34] Roach, A. *RTP Payload Format Restrictions*. RFC 8851. Jan. 2021. DOI: 10.17487/RFC8851. URL: https://www.rfc-editor.org/info/rfc8851.

[35] Lennox, J., Gross, K., Nandakumar, S., Salgueiro, G. and Burman, B. *A Taxonomy of Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources*. RFC 7656. Nov. 2015. DOI: 10.17487/RFC7656. URL: https://www.rfc-editor.org/info/rfc7656.

[36] Burman, B., Akram, A., Even, R. and Westerlund, M. *RTP Stream Pause and Resume*. RFC 7728. Feb. 2016. DOI: 10.17487/RFC7728. URL: https://www.rfc-editor.org/info/rfc7728.

[37] Burman, B., Wenger, S., Westerlund, M. and Chandra, U. *Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)*. RFC 5104. Feb. 2008. DOI: 10.17487/RFC5104. URL: https://www.rfc-editor.org/info/rfc5104.

[38] Burmeister, C., Rey, J., Sato, N., Ott, J. and Wenger, S. *Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)*. RFC 4585. July 2006. DOI: 10.17487/RFC4585. URL: https://www.rfc-editor.org/info/rfc4585.

[39] Hodson, O. and Perkins, C. *Options for Repair of Streaming Media*. RFC 2354. June 1998. DOI: 10.17487/RFC2354. URL: https://www.rfc-editor.org/info/rfc2354.

[40] Leon, D., Hakenberg, R., Rey, J., Miyazaki, A. and Varsa, V. *RTP Retransmission Payload Format*. RFC 4588. July 2006. DOI: 10.17487/RFC4588. URL: https://www.rfc-editor.org/info/rfc4588.

[41] Martin, J., Burbank, J., Kasch, W. and Mills, P. D. L. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. June 2010. DOI: 10.17487/RFC5905. URL: https://www.rfc-editor.org/info/rfc5905.

[42] Shankarkumar, V., Montini, L., Frost, T. and Dowd, G. *Precision Time Protocol Version 2 (PTPv2) Management Information Base*. RFC 8173. June 2017. DOI: 10.17487/RFC8173. URL: https://www.rfc-editor.org/info/rfc8173.

[43] Din, S. U., Ahmad, B., Ahmed, A., Amin, M. and Aoudi, S. Inter-destination Synchronization : A Comparison between Master-Slave and Synchronization-Manager Techniques. *2019 International Arab Conference on Information Technology (ACIT)*. 2019, pp. 222–229. DOI: 10.1109/ACIT47987.2019.8991020.

[44] Biersack, E., Geyer, W. and Bernhardt, C. Intra- and inter-stream synchronisation for stored multimedia streams. *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*. 1996, pp. 372–381. DOI: 10.1109/MMCS.1996.535000.

[45] Boronat, F., Lloret, J. and Garca, M. Multimedia group and inter-stream synchronization techniques: A comparative study. *Information Systems* 34.1 (Mar. 2009), pp. 108–131. DOI: 10.1016/j.is.2008.05.001. URL: https://doi.org/10.1016/j.is.2008.05.001.

[46] Li, Z., Uusitalo, M. A., Shariatmadari, H. and Singh, B. 5G URLLC: Design Challenges and System Concepts. *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*. 2018, pp. 1–6. DOI: 10.1109/ISWCS.2018.8491078.

[47] Schooler, E., Rosenberg, J., Schulzrinne, H., Johnston, A., Camarillo, G., Peterson, J., Sparks, R. and Handley, M. J. *SIP: Session Initiation Protocol*. RFC 3261. July 2002. DOI: 10.17487/RFC3261. URL: https://www.rfc-editor.org/info/rfc3261.

[48] *Ericsson Report 2022*. URL: https://www.ericsson.com/en/reports-and-papers/mobility-report/mobility-visualizer (visited on 09/14/2022).

[49] Poikselk, M., Holma, H., Hongisto, J., Kallio, J. and Toskala, A. *Voice over LTE (VoLTE)*. 1st. Wiley Publishing, 2012. ISBN: 1119951682.

[50] Association, G. *IMS Profile for Voice and SMS*. Apr. 2019. URL: https://www.gsma.com/newsroom/wp-content/uploads//IR.92-v13.0-2-1.pdf.

[51] *VoLTE Service Description and Implementation Guidelines, Version 1.1*. Mar. 2014. URL: https://www.gsma.com/futurenetworks/wp-content/uploads/2014/05/FCM.01-v1.1.pdf.

[52] Association, G. *Conferencing using the IP Multimedia (IM) Core Network (CN) subsystem. Version 17.0.0. Release 17*. Apr. 2022. URL: https://www.etsi.org/deliver/etsi_ts/124100_124199/124147/17.00.00_60/ts_124147v170000p.pdf.

[53] Camarillo, G. and Garca-Martn, M. A. *The 3G IP Multimedia Subsystem IMS*. Wiley, Aug. 2008. DOI: 10.1002/9780470695135. URL: https://doi.org/10.1002/9780470695135.

[54] Roach, A. *SIP-Specific Event Notification*. RFC 6665. July 2012. DOI: 10.17487/RFC6665. URL: https://www.rfc-editor.org/info/rfc6665.

[55] Rosenberg, J. *A Framework for Conferencing with the Session Initiation Protocol (SIP)*. RFC 4353. Feb. 2006. DOI: 10.17487/RFC4353. URL: https://www.rfc-editor.org/info/rfc4353.

[56] Schulzrinne, H., Rosenberg, J. and Levin, O. *A Session Initiation Protocol (SIP) Event Package for Conference State*. RFC 4575. Aug. 2006. DOI: 10.17487/RFC4575. URL: https://www.rfc-editor.org/info/rfc4575.

[57] Yang, M., Wang, S., Calheiros, R. N. and Yang, F. Survey on QoE Assessment Approach for Network Service. *IEEE Access* 6 (2018), pp. 48374–48390. DOI: 10.1109/ACCESS.2018.2867253.

[58] Suryanegara, M., Prasetyo, D. A., Andriyanto, F. and Hayati, N. A 5-Step Framework for Measuring the Quality of Experience (QoE) of Internet of Things (IoT) Services. *IEEE Access* 7 (2019), pp. 175779–175792. DOI: 10.1109/ACCESS.2019.2957341.

[59] Fizza, K., Banerjee, A., Mitra, K., Jayaraman, P. P., Ranjan, R., Patel, P. and Georgakopoulos, D. QoE in IoT: a vision, survey and future directions. *Discover Internet of Things* 1.1 (Feb. 2021). DOI: 10.1007/s43926-021-00006-7. URL: https://doi.org/10.1007/s43926-021-00006-7.

[60] J. S., S., M., V., V., P., V., M. and Panda, M. Sensor Data Harvesting Using an Autonomous Drone. *2020 5th International Conference on Communication and Electronics Systems (ICCES)*. 2020, pp. 612–616. DOI: 10.1109/ICCES48766.2020.9138075.

[61] Canetta, L., Mattei, G. and Guanziroli, A. Exploring commercial UAV market evolution from customer requirements elicitation to collaborative supply network management. *2017 International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. 2017, pp. 1016–1022. DOI: 10.1109/ICE.2017.8279993.

[62] Harris, M. The Radical Scope of Tesla's Data Hoard. *IEEE Spectrum* (2022).

[63] *Embedded Photo Metadata*. URL: https://developer.parrot.com/docs/pdraw/photo-metadata.html (visited on 08/19/2022).

[64] Kitanov, S. and Janevski, T. State of the art: Fog computing for 5G networks. *2016 24th Telecommunications Forum (TELFOR)*. 2016, pp. 1–4. DOI: 10.1109/TELFOR.2016.7818728.

[65] Rosenberg, J. *The Session Initiation Protocol (SIP) UPDATE Method*. RFC 3311. Oct. 2002. DOI: 10.17487/RFC3311. URL: https://www.rfc-editor.org/info/rfc3311.

[66] Mwanje, S. S. and Mannweiler, C. TOWARDS COGNITIVE AUTONOMOUS NETWORKS IN 5G. *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*. 2018, pp. 1–8. DOI: 10.23919/ITU-WT.2018.8597732.

[67] Okeme, P. A., Skakun, A. D. and Muzalevskii, A. R. Transformation of Factory to Smart Factory. *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*. 2021, pp. 1499–1503. DOI: 10.1109/ElConRus51938.2021.9396278.

[68] Corchero, C. and Sanmarti, M. Vehicle- to- Everything (V2X): Benefits and Barriers. *2018 15th International Conference on the European Energy Market (EEM)*. 2018, pp. 1–4. DOI: 10.1109/EEM.2018.8469875.

[69] Alalewi, A., Dayoub, I. and Cherkaoui, S. On 5G-V2X Use Cases and Enabling Technologies: A Comprehensive Survey. *IEEE Access* 9 (2021), pp. 107710–107737. DOI: 10.1109/ACCESS.2021.3100472.

[70] Lakkis, S. I. and Elshakankiri, M. IoT based emergency and operational services in medical care systems. *2017 Internet of Things Business Models, Users, and Networks*. 2017, pp. 1–5. DOI: 10.1109/CTTE.2017.8260983.

[71] Bisio, I., Fedeli, A., Lavagetto, F., Pastorino, M., Randazzo, A., Sciarrone, A. and Tavanti, E. Mobile Smart Helmet for Brain Stroke Early Detection through Neural Network-Based Signals Analysis. *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8255029.

[72] Lysogor, I. I., Voskov, L. S. and Efremov, S. G. Survey of data exchange formats for heterogeneous LPWAN-satellite IoT networks. *2018 Moscow Workshop on Electronic and Networking Technologies (MWENT)*. 2018, pp. 1–5. DOI: 10.1109/MWENT.2018.8337257.

[73] *IOTCOMMS.IO COMMUNICATIONS PLAFORM*. URL: https://iotcomms.io/ (visited on 09/26/2022).

[74] *SIPazon: The First AMS (Audio Management System) and Milestone for the Audio Industry*. URL: https://www.sipazon.com/ (visited on 09/26/2022).

[75] Portman, L., Rehor, K., Jain, R. and Hutton, A. *Use Cases and Requirements for SIP-Based Media Recording (SIPREC)*. RFC 6341. Aug. 2011. DOI: 10.17487/RFC6341. URL: https://www.rfc-editor.org/info/rfc6341.

[76] Hutton, A., Portman, L., Jain, R. and Rehor, K. *An Architecture for Media Recording Using the Session Initiation Protocol*. RFC 7245. May 2014. DOI: 10.17487/RFC7245. URL: https://www.rfc-editor.org/info/rfc7245.

[77] Shelby, Z., Hartke, K. and Bormann, C. *The Constrained Application Protocol (CoAP)*. RFC 7252. June 2014. DOI: 10.17487/RFC7252. URL: https://www.rfc-editor.org/info/rfc7252.

[78] Cirani, S., Davoli, L., Picone, M. and Veltri, L. Performance evaluation of a SIP-based constrained peer-to-peer overlay. 2014, pp. 432–435. DOI: 10.1109/HPCSim.2014.6903717.

[79] Association, G. *LTE-M Deployment Guide to Basic Feature Set Requirements*. June 2019. URL: https://www.gsma.com/iot/wp-content/uploads/2019/08/201906-GSMA-LTE-M-Deployment-Guide-v3.pdf.

[80] Dian, F. J. and Vahidnia, R. LTE IoT Technology Enhancements and Case Studies. *IEEE Consumer Electronics Magazine* 9.6 (2020), pp. 49–56. DOI: 10.1109/MCE.2020.2986834.

[81] Zhang, Z., Zhu, X., Li, Z. and Zeng, Y. Analysis of the Impact of eMTC on Legacy LTE. *2019 15th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2019, pp. 1133–1138. DOI: 10.1109/IWCMC.2019.8766733.

[82] U-Blox. *u-blox announces first LTE-M VoLTE call on European network infrastructure*. u-blox announces first LTE-M VoLTE call. 2019. URL: https://www.u-blox.com/en/press-releases/u-blox-announces-first-lte-m-volte-call-european-network-infrastructure (visited on 07/19/2022).

[83] Jones, P. and Hellstrom, G. *RTP Payload for Text Conversation*. RFC 4103. June 2005. DOI: 10.17487/RFC4103. URL: https://www.rfc-editor.org/info/rfc4103.

[84] *Sense HAT*. URL: https://datasheets.raspberrypi.com/sense-hat/sense-hat-product-brief.pdf (visited on 10/31/2022).

[85] Begen, A. C., Cai, Y. and Ou, H. *Duplication Delay Attribute in the Session Description Protocol*. RFC 7197. Apr. 2014. DOI: 10.17487/RFC7197. URL: https://www.rfc-editor.org/info/rfc7197.

[86] Brandenburg, R. van, Stokking, H., Deventer, O. van, Boronat, F., Montagud, M. and Gross, K. *Inter-Destination Media Synchronization (IDMS) Using the RTP Control Protocol (RTCP)*. RFC 7272. June 2014. DOI: 10.17487/RFC7272. URL: https://www.rfc-editor.org/info/rfc7272.

[87] Segu, F. B., Cebollada, J. C. G. and Mauri, J. L. An RTP/RTCP based approach for multimedia group and inter-stream synchronization. *Multimedia Tools and Applications* 40.2 (June 2008), pp. 285–319. DOI: 10.1007/s11042-008-0208-1. URL: https://doi.org/10.1007/s11042-008-0208-1.

[88] Montagud, M. and Boronat, F. Enhanced adaptive RTCP-based Inter-Destination Multimedia Synchronization approach for distributed applications. *Computer Net-*

works 56.12 (Aug. 2012), pp. 2912–2933. DOI: 10.1016/j.comnet.2012.05.003. URL: https://doi.org/10.1016/j.comnet.2012.05.003.

[89] Montagud, M., Boronat, F., Stokking, H. and Cesar, P. Design, development and assessment of control schemes for IDMS in a standardized RTCP-based solution. *Computer Networks* 70 (Sept. 2014), pp. 240–259. DOI: 10.1016/j.comnet.2014.06.004. URL: https://doi.org/10.1016/j.comnet.2014.06.004.

[90] Cox, C. *An Introduction to LTE: LTE, LTE-Advanced, SAE, VoLTE and 4G Mobile Communications*. 2nd. Wiley Publishing, 2014. ISBN: 1118818032.