sysmocom

systems for mobile communications GmbH

Ubuntu Touch VoLTE Investigation

Revision v1

July 22, 2021

Table of Contents

Table of Contents

1 Introduction

Ubuntu Touch is an alternative, fully Open Source operating system for smart phones. While its community has ported it to a variety of hardware devices, support for cellular network functionality is not complete. One particular missing gap is the support for VoLTE, which is a requirement in some markets today, while 2G and/or 3G networks are being phased out.

One device for which an Ubuntu Touch port exists is the “Volla Phone”. This phone is available with different operating systems. The Android OS on this phone provides VoLTE functionality, so we know

the hardware and modem firmware implementation is complete, only the application processor software is missing.

1.1 Existing Telephony Architectures

The two diagrams below illustrate the high-level architecture of components involved with telephony in the respective operating systems:

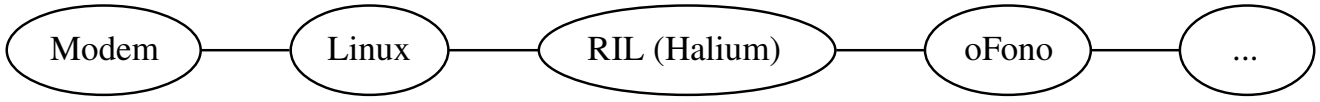


Figure 1: Telephony Architecture in Ubuntu Touch

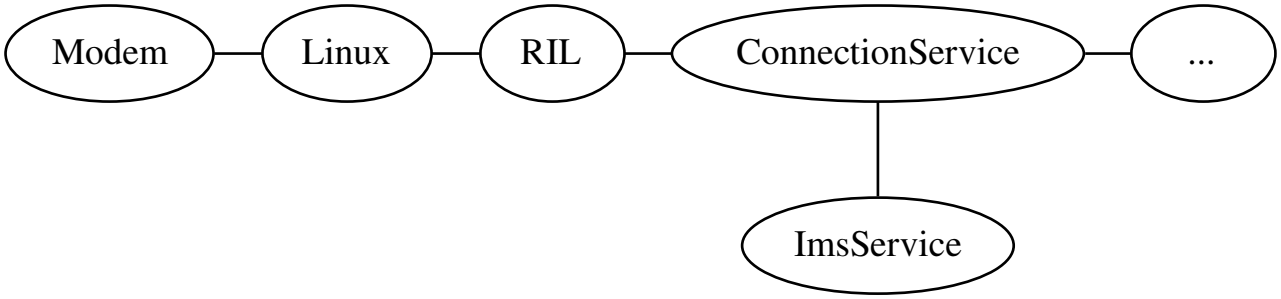


Figure 2: Telephony Architecture in Android

In order to come up with a strategy to support VoLTE on Ubuntu Touch, it is important to understand how VoLTE works in the [mostly proprietary] Android based VoLTE on identical hardware.

The RIL daemon (proprietary `mtkfusionrild` for the “Volla Phone”) is being accessed by different components in both operating systems (`oFono`, `ConnectionService`). The `ConnectionService` in Android is communicating with an `ImsService` (proprietary `com.mediatek.ims` for the “Volla Phone”) over the `ImsService` API. The `ImsService` implements the IP Multimedia Subsystem (IMS), on which VoLTE is based. Only the ImsService API is standardized by the Android platform. The implementation of that API is vendor specific and usually proprietary.

In Ubuntu Touch, IMS is so far not implemented.

1.2 Tracing of VoLTE Traffic

In order to propose a technological architecture, it is important to know which parts (if any) of the VoLTE traffic are routed through the application processor, and which parts are on the modem. We verified that in the case of the “Volla Phone” using `tcpdump` and `wireshark`:

- SIP/SDP traffic (for signaling) is handled on the application processor
- RTP traffic is absent, hence it must be handled on the modem processor

The same methodology can be used later during implementation of one of the proposed solutions, to compare the behavior with the original stack.

1.2.1 Running tcpdump in Android

We installed the latest Android build for the device, using the `ubports-installer` (and reported #1940 in the process, the installation had to be done twice to be successful). Root access was obtained with Magisk v22.1.

Archive name	volla-9.0-20201019-nightly-k63v2_64_bsp-signed.zip
Build number	PPR1.180610.011 dev-keys
Custom build version	alps-mp-p0.mp7-V2.51_prize.p0mp7.k57pv1.dm.64_P7

A familiar shell environment with `tcpdump` was set up by running the following inside Termux:

```
$ pkg update
$ pkg install openssh tsu tcpdump
$ passwd
$ sshd
```

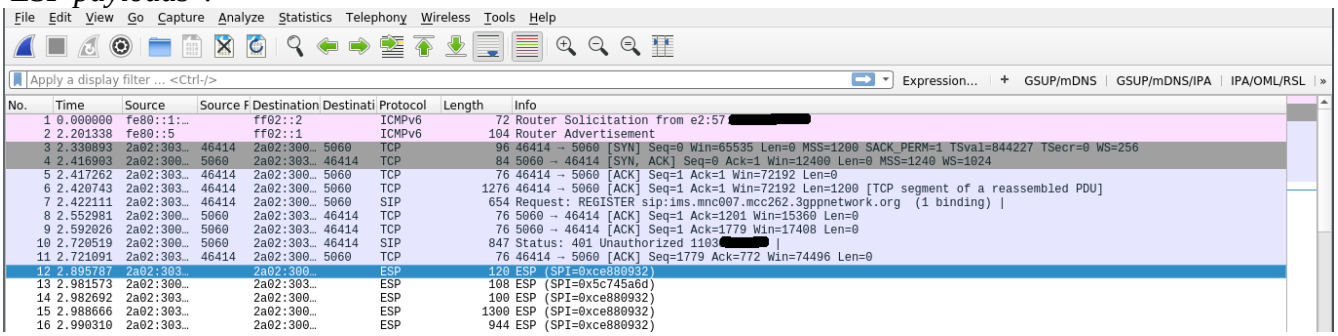
Then on the PC, to which the phone is connected via USB:

```
$ adb forward tcp:8022 tcp:8022
$ ssh -p 8022 localhost \
      sudo tcpdump -i any -s0 -w - -U -n not port 8022 \
      | wireshark -k -S -i -
```

In our testing, the interesting traffic was on interface (-i) `ccmni2`. But `any` can also be used since there is no interfering traffic.

1.2.2 Configuring Wireshark to show ESP payloads

In our testing we found that ESP is only used for integrity protection. Wireshark will not attempt to decode them by default (screen shot below), but can be configured to do so by opening the preferences, then selecting Protocols, ESP and ticking "*Attempt to detect/decode NULL encrypted ESP payloads*".



1.2.3 VoLTE registration

As the VoLTE connection gets established, SIP messages to REGISTER and SUBSCRIBE can be observed from the application processor to the operator IMS core:

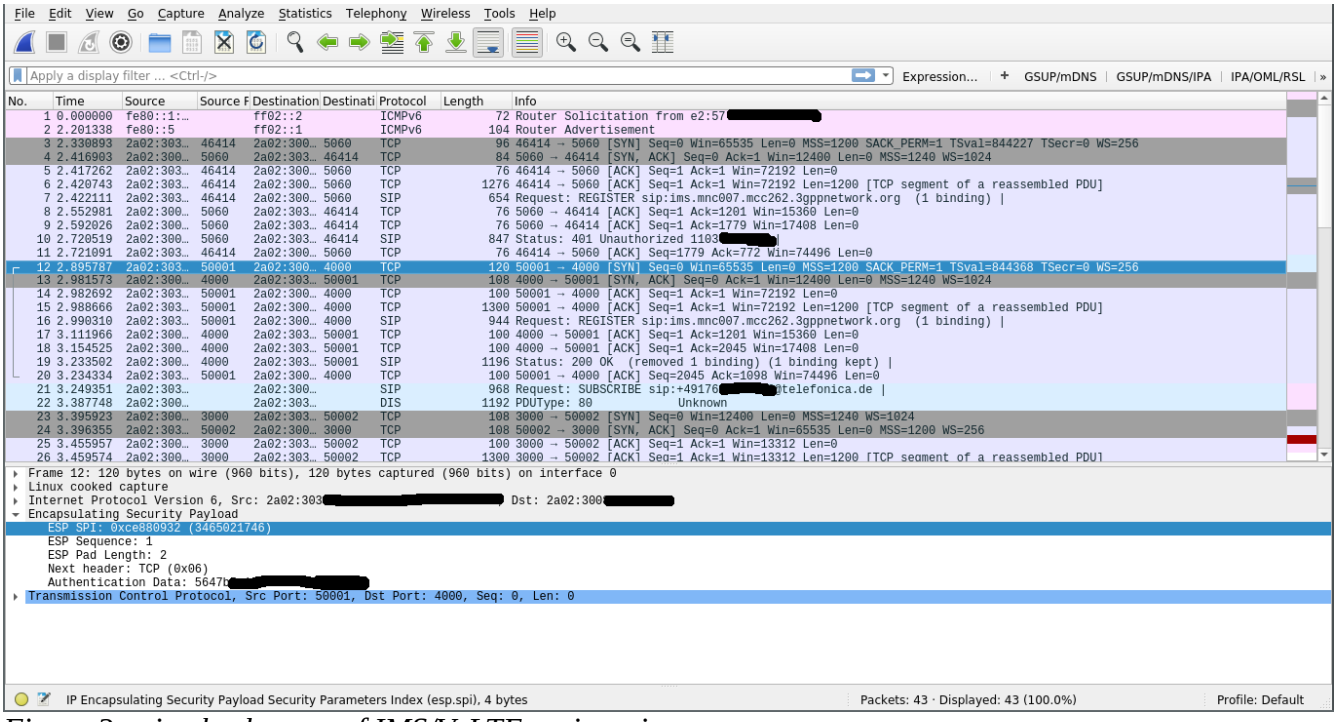


Figure 3: wireshark trace of IMS/VoLTE registration

1.2.4MT (mobile terminated) VoLTE call

The following screen shot shows data captured on the application processor during a call. It follows the SIP protocol with receiving an *INVITE* together with SDP-encapsulated information about the RTP stream, sending *Trying* and *Ringing* and waiting until the user answers the phone. Then the *OK* is sent, and an *ACK* is received.

The RTP traffic is not visible on the application processor, hence it must be completely handled by the modem.

When the user hangs up, a *BYE* is sent.

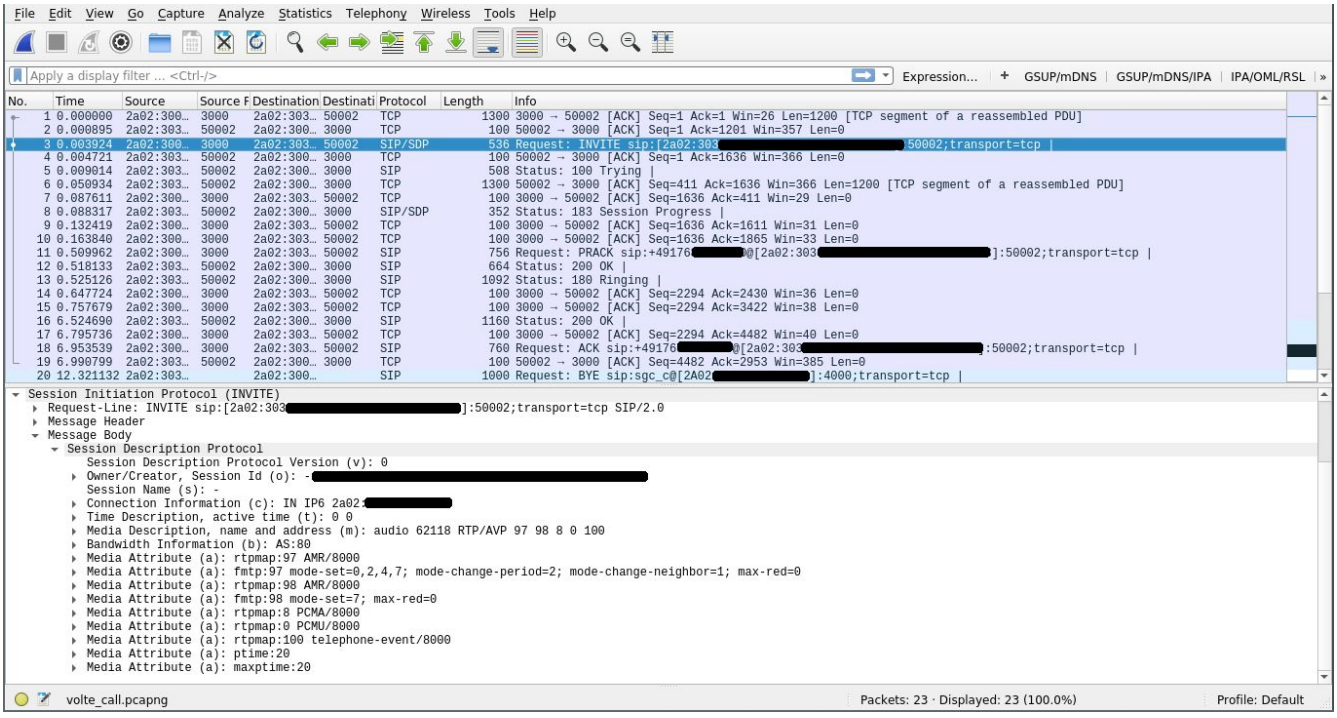


Figure 4: wireshark trace showing SIP signaling of TM VoLTE call

So all in all we can conclude that the entire SIP IMS signaling plane is implemented

2 Proposed Architecture for Ubuntu Touch

We see two alternatives in terms of a VoLTE architecture for Ubuntu Touch.

2.1 Open Source IMS Implementation

Run an open source IMS implementation in Ubuntu Touch, outside of Halium. This could either be implemented inside the oFono code base, or as standalone component which communicates with oFono (similar to the ConnectionService and ImsService architecture).

The oFono fork used in Ubuntu Touch is based on version 1.17, which does not have any IMS functionality. An IMS API was added to oFono 1.22 (Changelog, doc/ims-api.txt), but it only works with the Intel xmm7modem, and relies on the actual IMS implementation running in the modem and only being registered with AT commands from oFono (drivers/xmm7modem/ims.c). Therefore, it is not useful in the context of this project.

Despite VoLTE being completely based on publicly available standards by 3GPP and IETF, there is no existing, maintained open source IMS client implementation available. This is even more surprising as there was a lot of activity in the 2008 to 2012 timeframe, when IMS was still new and various research projects were built around it. Those projects all were aimed at the actual IMS itself, and not at the LTE specific implementation of IMS as used in VoLTE.

A 2012 summary of the state of existing, but now abandoned/defunct open source IMS client implementations can be found at https://nil.uniza.sk/six-open-source-ims-clients-features-overview-february-2012/ - the only native Linux one in that list is UCT IMS Client, abandoned in about 2012.

A brief analysis of those code bases showed that the doubango framework looks like the most complete and reusable codebase among the available options. There are three existing FOSS IMS clients based on the doubango framework: imsdroid (Android), idoubs (iOS), boghe (Windows Phone 8). A native Linux client is unfortunately missing.

The goal is to replace all required functionality of the proprietary ImsService. However, there is great variation in what the ImsService is doing depending on the vendor's implementation. For com.mediatek.ims, we found that it is implementing IPsec and VoIP signaling (on the application processor)(on the application processor). From other modems, such as the Quectel EG25-G, we know that IPSec and VoIP are handled on the modem processor.

In this architecture proposal, we recommend adapting and enhancing an open source IMS implementation (doubango) for VoLTE, with platform/vendor specific support on the Volla phone. A proof of concept should be possible by comparing traces of this new implementation with traces of the existing implementation (e.g. between RIL/ConnectionService, ConnectionService/ImsService and on the network interfaces). In the case of "Volla Phone" at least with one German operator, we found that the IPSec ESP traffic is not encrypted and can be viewed in Wireshark. The implementation could be made more robust by testing it in a private VoLTE network, which allows testing corner cases and tracing from the core network side.

If the first implementation was done to be compatible with a Mediatek device, it is likely that it will work for other Mediatek devices too with either no or few adjustments. Once the process is done for one vendor, it should be easier to extend the open source IMS implementation to work with another phone/modem vendor.

2.2 Proprietary IMS Service in Halium

An alternative proposal is running the proprietary `ImsService` inside Halium. The benefit is, that once the whole ecosystem to run the `ImsService` is in place, it should in theory be possible to run the `ImsService` implementation of any vendor.

In contrast to the RIL daemon and related binaries, which are already running inside Halium, the `ImsService` is always implemented in Java. It is currently not possible to run such Java-based services inside Halium. So this would need to be implemented in Halium first. We do not have much experience with Halium, but Florian Leeber noted that this is a non-trivial larger development effort, and that it may lead to requiring a bigger system partition and therefore problems with older devices.

Once the proprietary `ImsService` is running inside Halium, oFono can be extended to communicate with `ImsService` through Android's `ImsService` API. During development of the oFono improvements, traces and a private VoLTE network could be utilized to compare with Android and to verify that everything is working correctly (as in 2.1).

3 Recommendation

While proposal 2.2 has the attractive potential of making `ImsService` implementations of any vendor work, this is still an assumption that can only be verified once a large effort of development has been invested. We recommend to rather invest these efforts in proposal 2.1. Even though the result will likely be more vendor specific, it yields a full open source IMS stack + application, which is more efficient and maintainable in the long run.

As IMS is also used for VoWiFi, a FOSS IMS client developed within proposal 2.1 would almost immediately also enable VoWiFi use cases: VoWiFi effectively is the same IMS over IPsec over the public internet, instead of the LTE related dedicated bearers.

4 Proposed Project Plan

The high-level goal of the project plan is to have functional IMS voice calls over LTE on the Volla Phone. sysmocom would take care of implementing anything up to the point where this can be demonstrated with a simple command-line client on the target device, but not work on UI / application exposure of the VoLTE call functionality.

4.1 Milestones

The first set of milestones are labeled “Desktop”. This means that development happens on a normal “desktop” GNU/Linux system, and not yet on the target hardware (Volla Phone). This results in a more rapid development cycle, as all parts of the software not strictly related to the target hardware and OS environment can be debugged natively on the desktop PC.

Later on, once everything is working on the desktop PC, an interoperable IMS implementation exists, and that known-working software is integrated with the target environment. Hence, related milestones are subsequently running on the target device.

4.1.1 Desktop: IPsec towards commercial operator

Capability to successfully establish an IPsec session to the operator core network, using EAP-AKA and the ISIM for authentication.

4.1.2 Desktop: IMS registration to commercial operator

Capability to perform a successful IMS registration to the operator core network, including authentication using the ISIM. At this point, IMS signaling (e.g. incoming MT calls) from the operator CN arrives at the client, but is not handled yet.

4.1.3 Desktop: MO + MT voice call IMS signaling

Capability to handle signaling related to MO and MT voice calls. That means, successful handling of all normal phases of a call, until its release. No voice audio is yet included.

4.1.4 Desktop: voice call user plane

Capability to perform voice calls with working user plane (via sound card present in desktop PC)

4.1.5 Volla: IMS signaling via QCI=5 dedicated bearer

This resembles milestone 5.1.2 / 5.1.3, but not on the desktop PC over the public internet but on the target device via the dedicated radio bearer on QCI=5 over LTE. It includes any required interaction with the cellular modem in the target device to create (and later remove) that QCI=5 bearer.

4.1.6 Volla: IMS user plane via QCI=1 dedicated bearer

This resembles milestone , but not on the desktop PC over the public internet but on the target device via the dedicated radio bearer on QCI=1 over LTE. It includes any required interaction with the cellular modem in the target device to create (and later remove) that QCI=1 bearer, as well as handling

the RTP voice audio handling (encoding/decoding, playing voice and recording voice from the microphone).

4.2 List of major tasks

Taking up proposal 3.1 would require the following major work items:

1. IMS interoperability of doubango against commercial cellular network
    1.1. P-CSCF and/or ePDG resolution / discovery procedures
    1.2. Integration of ISIM for authentication on SIP/IMS domain
    1.3. Integration of Linux kernel IPsec (IKE in doubango or extending StrongSwan)
    1.4. Whatever missing bits required for interoperability of IMS registration
    1.5. Whatever missing bits required for MO Voice Call Signaling
    1.6. Whatever missing bits required for MT Voice Call Signaling
    1.7. Whatever missing bits required for Voice Call with User Plane
2. doubango IMS / LTE Integration
    2.1. Hooks for platform specific code to trigger the creation of dedicated bearers for QCI=1 and QCI=5
3. Platform specific integration
    3.1. Establishing dedicated LTE bearer for QCI=1 (IMS signaling)
    3.2. Establishing dedicated LTE bearer for QCI=5 (IMS RTP user plane)
    3.3. Instructing modem to handle RTP flow? How to tell it about the IP/Port/codec/...?

4.3 Out of scope

In order to not constrain the scope of this initial project, we currently intentionally declare the following topics out of scope:

1. Supplementary Services (like call waiting, multiparty, hold and retrieve, etc.)
2. VoNR functionality (voice over 5G New Radio)
3. SRVCC (seamless hand-over between 2G/3G CS call and 4G/IMS call)

4. SMS over IMS (most operators should use SMS-over-SGs anyway)
5. RCS (Rich Communication Suite)
6. UI related work; integration into the Dialer/Phone application on UT

Exclusion of the above topics from the scope does not prevent any of them to be added later on to the resulting software.

4.3.1 UI Integration

As stated above, UI integration is out of scope for the proposed project.

As there are already three existing IMS client applications using the doubango IMS framework, it is assumed that whatever doubango provides in terms of APIs is sufficient for developing related client software.

sysmocoms key specialty is within telecom protocols and not UI - specifically we have no prior knowledge or experience implementing UI on Ubuntu Touch, particularly not the existing UI software used for making circuit-switched voice calls.

4.4 Development Process

The development happens as true open source project, with the following considerations:

- all development happens in public/open git repositories, no private/internal repos
- all features and bugs are tracked in a public issue tracker
- meaningful commit log messages
- one feature per commit
- attempt to contribute any related changes with upstream (might be challenging in case of doubango which has not seen a lot of activity in recent years)
- interact with the wider community, accept and integrate any related contributions. sysmocom does not try to maximize its involvement, but is happy for any part of the project that is resolved by the wider Ubuntu Touch community.

5 Project Phases / Effort

It is suggested to split the project in two phases.

5.1 Phase 1: Exploration and Prototyping

- On Desktop PC: Get more experience with doubango; try to get it to establish IMS signaling with a commercial operator via IPsec and EAP-AKA with ISIM. This allows us to get a better view into what magnitude of gaps or interop problems there are in doubango regarding the use in a commercial IMS network
- On Volla Phone: Investigate thoroughly the interaction betwen ImsService+ril <-> modem regarding the establishment of decicated bearers and the activation of RTP voice handling on the modem

At the end of those phase, we should be able to refine the plan and have a better idea regarding the effort of Phase 2. We will also know if we are hitting major roadblocks regarding the details of the Volla modem interaction, which could jeopardize the entire project. In the worst case, the project could be stopped at this point, before the majority of the effort is invested in developing the full-blown implemetation.

Phase 1 itself is estimated at 15 person-days.

Phase 2: Implementation

Use the results from Phase 1, the actual full-blown implementation of whatever is needed to reach the goals.

It is currently too early to give a realistic estimate of the amount of effort. An early “gut feel” guesstimate would be in the order of magnitude of 40 person-days. A lot depends on the size of the gaps discovered in doubango during Phase 1.