Linux Telephony

A Short Overview

Paul P. Komkoff
Google Ireland Ltd.
i@stingr.net

Anna Anikina
Saratov State University
anna@sgu.ru

Roman Zhnichkov
Saratov State University
zr@sgu.ru

Abstract

This paper covers the topic of implementing voice services in packet-switched Voice Over IP (VoIP) and circuit-switched (traditional telephony) networks using Linux and commodity hardware. It contains a short introduction into general telephony, VoIP, and Public Switched Telephony Networks (PSTN). It also provides an overview of VoIP services, including the open-source software packages used to implement them, and the kernel interfaces they include. It explains kernel support for connecting Public Switched Telephony Networks using digital interfaces (E1/T1) via the *Zaptel* framework, and user-space integration issues. The conclusion examines current trends in Linux-based and open-source telephony.

A basic understanding of networking concepts is helpful for understanding this presentation.

1 Introduction to general telephony, VoIP and PSTN

Although VoIP products like *Skype*™ and *Google Talk*™ are storming the telecommunications landscape, almost everyone still owns a telephone and uses it daily. Understanding how telephony works is still an obscure topic for most people. As a result, we would like to begin our paper with a short introduction to telephony in general, combined with two major types presently in use—packet-switched and circuit-switched.

Every telephony system needs to transmit data between parties, but before it can do so, it needs to find the other party and a route to it. This is called *call setup*. Activity of this nature is usually performed by a *signalling protocol*. Data can be passed via the same route and channel as signalling, or via the same route and a different channel, or via an entirely different route. There are obvious *quality of service* (QOS) requirements for signalling and data—signalling requires guaranteed delivery of every message, and data requires low-latency transmission, but can lose individual samples/frames.

1.1 VoIP

Voice over IP (VoIP) is widely used to describe various services, setups, and protocols that pass audio data in pseudo real-time over IP networks. Although the actual implementations are very different, the basic requirements are to pass voice data in two directions and to allow two-party conversation. VoIP is *packet-switched telephony* because the underlying network is packet-switched. To make conversations possible a virtual circuit is built between parties.

There are many protocols used for VoIP conversations. The most widespread is Session Initiation Protocol (SIP). This is a signalling protocol in that it only handles call setup and termination. Actual session setup between endpoints is handled by *SDP* (Session Description Protocol), and data is transmitted by *RTP* (Real-Time Protocol). SIP is described in RFC3561 and endorsed by IETF as an Internet standard.

Another protocol with a large user-base is *H.323*. While SIP is designed mostly by network people it is similar to HTTP (its messages are text `Name:Value` pairs). H.323, being endorsed by *ITU*—telephony people—looks like a traditional telephony protocol. Its messages and information elements are described in *ASN.1* (Abstract Syntax Notation) and coded by ASN.1 *BER* (Basic Encoding Rules). H.323 is also a signalling protocol, and also uses RTP for data transmission.

Inter-Asterisk Exchange (IAX) is another major VoIP protocol. It was invented by the *Asterisk*® authors. In this protocol, data and signalling streams are not separated, which allows easier NAT traversal. IAX is also able to pack multiple conversations into a single stream, thus reducing trunking overhead. There are two versions of IAX, although IAX1 is deprecated, and IAX2 is used under name of IAX.

Other lesser known VoIP protocols are either proprietary (*SCCP* is used in Cisco phones) or designed for specific purpose (*MGCP*—Media Gateway Control Protocol—used to manipulate VoIP gateways). Some protocols allow direct conversation between *endpoints* while others require a server to operate.

VoIP protocols define the way voice data should be transmitted. Conversion of digitized audio into *payload* (portions of data to be delivered by the VoIP protocol) is performed by *voice codecs*. The most popular voice codecs are: G.711, G.723, G.726-G.729, iLBC, GSM06.10, and Speex. Each has different bandwidth requirements, CPU requirements, quality, and patents associated with them. These codecs are not all VoIP-specific—G.711 is used in traditional telephony, and GSM06.10 is used in GSM mobile networks.

1.2 PSTN

Public Switched Telephony Network (PSTN) is the traditional telephony network. Capable nodes in this network are addressed using global and unique *E.164 addresses*—telephone numbers. Nowadays PSTN includes not only traditional telephones, but mobile phones as well. Most of the PSTN is digital (except customer analog lines and very old installments in developing countries). PSTN is *circuit-switched*, which means that the call setup procedure assembles a circuit between the parties on the call for the duration of the entire call. The circuit is either fixed-bandwidth digital (typically 64kbps for traditional telephone networks and 9600bps for mobile networks) or analog—spanning multiple units of communication equipment. In digital networks the circuit is made over *E1* (for Europe) or *T1* (America) lines, which contain 31 or 24 DS0 (64kbps) circuits, TDM-multiplexed together and thus often called *timeslots*.

The call management, call routing, circuit assignment and maintenance procedures are performed by the *signaling protocol*. The de facto standard for interconnecting networks is Signaling System 7 (SS7). Connections to customer *PBX* (Private Branch Exchange) are often performed using ISDN PRI (Primary Rate Interface) connections and ISDN Q.931 signaling. SS7 is not a single protocol, but a protocol stack. It contains *parts* which facilitate running SS7 itself and allows *user parts* to run on top of it. The user part that is responsible for voice call setup between parties is called *ISUP* (ISDN User Part). Services for mobile subscribers, ranging from registration to SMS, are handled by *MAP* over *TCAP* (Mobile Application Part over Transaction Capabilities Application Part) of SS7.

2 Implementing VoIP services on Linux

To implement any voice service using VoIP, we do not need any special hardware. Both clients and servers are done in software. We only need to implement a particular VoIP protocol (SIP, H.323, IAX, MGCP, SCCP) and a particular set of voice codecs.

After we have everything ready on the protocol and codec sides, we can implement the voice service. For example, we can build a server that will handle client registrations and calls to each other. This piece of software is typically called a *softswitch* because it functions much like a hardware switch—building virtual circuits between parties. Softswitches typically have the ability to provide optional services commonly found in traditional proprietary PBXes like conferencing, voicemail, and *IVR* (Interactive Voice Responce) for an additional charge. Modern opensource VoIP software logic is driven by powerful scripting languages—domain-specific (for building dialplans) or general purpose. This allows us to integrate with almost anything. For example, we can try opensource speech synthesis/recognition software.

Many softswitches utilize special properties of particular VoIP protocols. For example, SIP and the H.323 architecture provide the ability to pass data directly between endpoints to reduce contention and minimize latency. Thousands of endpoints can be registered to one SIP server to control only signalling, allowing billing and additional services. This is much better than sitting in the middle of RTP streams between those clients. Moreover, sometimes it is possible to pass data directly between two endpoints while one of them is using SIP and another—H.323. This setup is called a *signalling proxy*.

Some softswitches are suitable only for VoIP clients (including VoIP-PSTN gateways acting as VoIP endpoint) while more general solutions are able to act as a switch between different VoIP protocols and PSTN itself. The softswitches of the first kind are, for example, `SIP Express Router` and `sipX`, while the major players of the second kind are: `Asterisk`, `CallWeaver` (described later on), `YATE`, and `FreeSwitch`.

To test the most widespread softswitch—`Asterisk`—using VoIP only, download its source code from `http://asterisk.org`, compile it, and install. Without any additional software you have out-of-the-box support for SIP and IAX, a working IVR demo (in `extensions.conf` file), and many functions which you can attach to numbers in extensions.conf—*asterisk applications*. However, conferencing won't work and music-on-hold can stutter.

Call-center solutions based on Asterisk usually utilize `Queue()` application. Using different *AGI* (Asterisk Gateway Interface) scripts and builtin applications like `SayNumber()`, you can build an automatic answering machine which reports the current time or account balance. Asterisk can make outgoing calls as well if a specifically formatted text-file is added to the special directory for each call.

Another software package to try is Yate. Its architecture is different, however, you still can easily test basic functions of an IP PBX. Yate can be configured to be a signalling proxy between H.323 and SIP—a desired usage when building VoIP exchanges.

What does *Linux support for VoIP* mean here? It means fast, capable UDP (and ioctls which permit setting a particular DSCP on outgoing packets), a CPU scheduler which will not starve us receiving (if we are using blocking/threaded model), sending, and processing, and a preemptive kernel to reduce receive latency. However, there are still problems when a large number of clients are passing data through a single server.

Recent improvements in the kernel, namely in the scheduling, preemption, and high-precision timers have greatly improved its ability to run userspace telephony applications.

2.1 VoIP clients

There are two primary types of VoIP clients or *endpoints*—those running on dedicated hardware (handsets plugged into Ethernet, analog telephone adapters, dedicated VoIP gateways), and softphones—installable applications for your favorite operating system.

Popular open source softphones include: `Ekiga` (previously Gnome Meeting), `Kphone`, and `Kiax`, which support major protocols (H.323, SIP, and IAX). The supported voice codecs list is not as long as it might be due to patent issues. Even with access to an entirely free alternative like Speex, the user is forced to use patented codecs to connect to proprietary VoIP gateways and consumer devices.

Skype™, a very popular proprietary softphone, implements its own proprietary VoIP protocol.

3 Connecting to PSTN

In order to allow PSTN users to use the services described above, or at a minimum send and receive calls from other VoIP users, they need to connect to the PSTN. There are several ways to do that:

- analog connection, either FXO (office) or FXS (station) side
- ISDN BRI (Basic Rate Interface) connection
- ISDN PRI or SS7 on E1/T1 line

We will concentrate on the most capable way to connect a E1/T1 digital interface (supporting ISDN PRI or SS7 directly) to a VoIP server. Carrier equipment is interconnected in this way. E1/T1-capable hardware and kernel is required to support this.

The "original" digital telephony interface cards compatible with Asterisk are manufactured by Digium®. Each contains up to 4 E1/T1/J1 ports. Other manufacturers have also unveiled similar cards, namely Sangoma and Cronyx. Clones of the Digium cards are also available in the wild (OpenVox) which behave in exactly the same way as the Digium ones.

One way to present telephony interfaces to an application is by using the *Zaptel* framework. The official zaptel package, released by Digium together with Asterisk, contains the zaptel framework and drivers for Digium and Digium-endorsed hardware. Although drivers for other mentioned hardware have different architectures, they implement zaptel hooks and are thus compatible (to a certain extent) with the largest user base of such equipment. However, other software can use other ways of interacting with hardware. For example, Yate (partially sponsored by Sangoma), can use native ways of communicating with Sangoma cards. Extremely high performance has been shown in those setups.

After you have ISDN PRI provisioned to you and a Digium card at hand, obtain the latest Zaptel drivers (also at `http://asterisk.org`) and compile them. If everything goes well and you successfully insmod (load) the right modules (and you have udev), you will notice a couple of new device nodes under `/dev/zap`. Before starting any telephony application, you need to configure zaptel ports using `ztcfg` tool. After configuration you will have additional nodes `/dev/zap/X`, one for each channel you configured. In ISDN PRI, the 16th timeslot of E1 is dedicated signalling channel (D-chan). As a result it runs Q.931 over Q.921 over HDLC. All other timeslots are clear-channels (B-chan) and are used to transmit data. At a minimum, the ISDN PRI application needs to talk Q.931 over the D-channel, negotiate B-channel number for conversations, and read/write digitized audio data from/to the specific B-channel.

Achieving SS7 connectivity is slightly more difficult. Until 2006, there was no working opensource SS7 implementation. Even today, you still need to find a carrier who will allow an uncertified SS7 device on their network. On the other hand, when you are *the* carrier, having opensource SS7 is extremely useful for a number of reasons. One might be your traditional PSTN switch—which has only SS7 ports free when buying ISDN PRI ports isn't an option.

Today there is at least one usable SS7 implementation for asterisk—Sifira's `chan_ss7`, available at `http://www.sifira.dk/chan-ss7/`. An opensource SS7 stack for Yate (yss7) is in progress.

What kind of services can we implement here? VoIP-PSTN gateway? Indeed, if we are able to capture the voice onto the system, we can transmit and receive it over the network. Because we use an opensource softswitch for this purpose, we get a full-fledged IP-PBX with PSTN interface, capable of registering softphones and hardphones and intelligently routing calls between VoIP and PSTN. This also includes call-center, IVR, and Voicemail out-of-the-box, and is flexible enough to add custom logic. If our network requires multiple such gateways, we can replicate some of the extra functionality between them and setup call routing in a way that eliminates unneeded voice transfers over the network, thus reducing latency.

The described setup can also be used as a dedicated PSTN system. With this method, you can still use the networking features if your setup consists of more than one node—for configuration data failover or bridging of calls terminated on different nodes.

Advanced usage scenarios for hardware with single or multiple E1 interfaces are usually targeted for signalling. If we take a card with 2 E1 interfaces, cross-connect together all the timeslots except 16 from port 1 to port 2, and then run an application which speaks Q.931 on timeslot 16 of port 1, and transparently translate it to SS7 ISUP on timeslot 16 of port 2, we will have a *signalling converter*. This is used to connect ISDN-only equipment to a SS7-only switch. If we implement SS7 TCAP/MAP, we can create a SMS center out of the same hardware or build IN SCP (Intelligent Network Service Control Point).

Although the E1/T1 connection option is used in the majority of large-scale voice services, you may still need an analog or ISDN BRI line to connect to your server. Digium and others vendors offer analog and ISDN BRI cards which also fit into the Zaptel framework.

3.1 Echo

When interconnecting VoIP and PSTN it is not uncommon to have problems with echo. *Hybrid transition* refers to the situation where the incoming and outgoing signals are passed via a single 2-wire line and separated afterwards, thereby reflecting some of the outgoing signal back. It is also possible for analog phones or headsets to "leak" audio from headphones or speakers to the microphone. Circuit-switched phone networks are very fast and as a result echo is not noticeable. This is because there are two digital-analog conversions on each side and digitized audio is passed in single-byte granularity resulting in low latency. VoIP installations which include voice codecs (adding more overhead) and passing multiple samples in one packet, may introduce enough latency to result in noticable echo.

To eliminate echo, echo cancellation can be added which subtracts the remnants of the outgoing signal from the incoming channel, thus separating them. It is worth mentioning, however, that if in an `A-B` conversation, party A hears an echo, there is nothing you can do on the A side—the problem (unbalanced hybrid, audio leak, broken echo canceller) is on the B side.

3.2 Fax over IP transmission

Faxing over VoIP networks does not work for a number of reasons. First, voice codecs used to reproduce voice data are *lossy*, which confuses faxmodems. Using G.711 can be lossless if you are connected using a digital interface. This is because when used in PSTN DS0 circuits, the unmodified data is passed as the payload to the VoIP protocol. If you do this, however, then echo cancellation will still mangle the signal to the point that modems cannot deal with. As a result, you also need to turn off echo cancellation. Unfortunately, this means the internal modem echo-canceller will not be able to deal with VoIP echo and jitter to the point where it will not work.

There are a number of solutions for this problem. The first is called *T.37—store and forward* protocol. With store and forward, the VoIP gateway on sending end captures the fax and transmits it to the gateway on the receiving side using SMTP. The second method is *T.38*—which tries to emulate *realtime* fax behavior. This is usually more convenient when you send faxes in the middle of the conversation.

4 Zaptel architecture

Most digital interface cards come with one or a combination of interfaces, which together, form a *span*. Data in G.703 E1 stream travels continuously, but cards are usually programmed to signal an interrupt after a predetermined configured amount of data is receiveed in a buffer. In addition, the interrupt is generated when data transmission is finished.

The Zaptel hardware driver provides the following functionality:

- empty data from the device receive buffer, rearrange it channelwise (if needed) and fill the zaptel receive buffer
- call echo cancellation hooks and call `zt_receive(&p->span)`—on the receive path
- call `zt_transmit(&p->span)`, call echo cancellation hooks, empty data of zaptel transmit buffer, rearrange it channelwise and put into device from the transmit buffer, and queue transmission—on the transmit path.

This basic API makes writing drivers very easy. Advanced features can also be implemented too. For example, some E1 cards have the ability to *cross-connect* timeslots without passing the data to the software—useful when both parties are sharing channels of the same span, or for special telephony applications. This feature can be implemented (with some effort) and is supported by current versions of Asterisk.

Clear channels, used for voice data, are passed to userspace unmodified. *Signalling channels*, however, need modification performed by the Zaptel level. ISDN signalling (Q.931 on top of Q.921) requires HDLC framing in the channel, which must be implemented in the kernel. The `ztcfg` tool is used to configure the channel as `D-chan`.

While HDLC framing is done at the kernel-level, Q.931 signalling itself must be done in userspace. Digium offers a library (libpri) for this. This driver was originally used in the zaptel channel driver in asterisk—used now in most ISDN-capable software. SS7 signalling is slightly more difficult as it requires continuous *Fill-In Signal Unit* (FISU) generation which must be placed in the kernel (at the zaptel level) for reliability.

4.1 Code quality issues

Although the zaptel-related hardware driver part seems straighforward, zaptel itself isn't that good. Its 200-kilobyte, 7,000 line single source file includes everything plus the kitchen sink which Asterisk depends heavily on. Due to the continuous flow of data, zaptel devices are often used a as stable source of timing, particularly in the IAX trunking implementation and for playing Music-On-Hold to VoIP clients. To use this feature without Zaptel hardware you need the special `ztdummy` driver which uses RTC and emulates the zaptel timing interface. Also, for reasons we cannot explain, the zaptel kernel module contains a user-space API for conferencing. This module allows the attachment of multiple readers/writers to a particular device node and does all mixing in kernel space. Thus, to enable asterisk conferencing, you also need zaptel hardware or ztdummy. Echo cancellation is selectable and configurable only at compile-time. This is inconvenient when troubleshooting echo problems.

Consistent with every external kernel module that is supposed to work with 2.4 and 2.6 kernels, zaptel contains lots of `#ifdefs` and wrapper macros. It is unclear if Digium will ever try to push Zaptel to mainline—in its current state we think that is impossible.

4.2 Cost, scalability and reliability

Most telco equipment is overpriced. Although we have found PBXes with an E1 port and 30 customer ports for a reasonable price, the base feature set is often very limited. Each additional feature costs additional money and you still will not receive the level of flexibility provided by open-source software packages. Options for interconnecting PSTN and IP are even more expensive.

Telco equipment is overpriced for a number of reasons—mostly due to reliability and scalability. By building a telco system out of commodity hardware, the only expensive part is the E1 digital interface. Even with this part we are able to keep the cost of single unit low enough to allow $1+1$ (or even $1+1+1 spare$) configuration, and the price of hardware will still be much lower. This approach allows us to reach an even higher level of reliability than simply having one telephony switch. This is because we can take units down for maintenance one-by-one.

Combining different existing solutions also reduces some limitations. For example, if the number of VoIP clients in our VoIP-PBX with PSTN connection is so high that asterisk cannot handle the load, we can put a lightweight SIP proxy (OpenSER) in front of it, and all internal VoIP calls will close there.

4.3 Performance issues

There are some inefficiencies in PSTN processing from a performance point of view, which are dictated by the Zaptel architecture. Some cards generate interrupts for each port. For example, with a sample length of 1ms (`ZT_CHUNKSIZE == 8`) there will be 1,000 interrupts per second per port. If we add a large number of ports in a single machine, this number will be multiplied accordingly. There are ways to reduce interrupt load. For example, the card can generate a single interrupt for all its ports. Another way is to use larger samples, but this introduces significant latency and is thus discouraged.

Another zaptel problem is that it creates individual device nodes for every channel it handles. Although with recent kernels, we can easily handle lots of minors, reading from individual channels just does not scale. This can be optimized by feeding all channels via a single device node—but we need to be careful here, because there will be signalling in some timeslots. Also, echo cancellation and DTMF detection can double CPU load. Offloading them to dedicated hardware can save 50% of CPU time.

Better performance can also be achieved by simplifying the hardware driver architecture by eliminating complex processing—echo cancellation or DTMF detection—in the kernel (or interrupt context) by coupling clear channels together before feeding them to userspace. Echo cancellation can be performed on hardware or software—in userspace. However, using software echo cancellation and DTMF detection can be more cost-effective—compare the cost of adding another CPU vs. the cost of hardware EC/DTMF detectors.

However, using more servers with less E1 ports may be wise from a reliability point of view. Modern CPUs have enough processing power to drive 4 E1 interfaces even with a totally unoptimized zaptel stack and userspace. Thus, for large setups we can have any number of 4-port servers connected to a high-speed network. If we are interconnecting with VoIP clients here, we can split the load across the 4-port nodes, and the maximum number of VoIP clients will be no more than 120.

5 Current trends

Until recently, Asterisk dominated the opensource telephony landscape. Zaptel and Asterisk were directed by Digium which sells its own telephony hardware. Recently, however, other players stepped up both on the hardware and software fronts.

Sangoma Technologies, a long time procucer of E1/T1 cards, modified its WANPIPE drivers to support Zaptel. Cronyx Engineering's new drivers package also includes the zaptel protocol module.

There are three issues in the Asterisk universe which resulted in the forking of OpenPBX, later renamed to CallWeaver. Those issues are:

1. Requirement to disclaim all copyrights to Digium on code submission, due to Asterisk dual-licensing and Digium commercial offerings.

2. Because of dual licensing, Asterisk is not dependent on modern software libraries. Instead, it contains embedded (dated) Berkeley DB version 1 for internal storage.

3. Strict Digium control on what changes go into the software.

CallWeaver was forked from Asterisk 1.2 and its development is progressing very rapidly. Less than a year ago they switched from a fragile custom build system to automake, from zaptel timing to POSIX timers, from zaptel conferencing to a userspace mixing engine, from internal DSP functions to Steve Underwood's SpanDSP library, and from awkward db1 to flexible SQLite. CallWeaver has working T.38 support, and is still compatible with zaptel hardware. CallWeaver developers are also trying to fix architectural flaws in Asterisk by allowing proper modularization and changing internal storage from linked lists to hash tables.

Although CallWeaver contains many improvements over Asterisk, it still shares its PBX core, which was designed around some PSTN assumptions. For example, it is assumed that audio data is sampled at 8khz. This is good for pure PSTN applications (or PSTN/VoIP gatewaying), but in VoIP environments we might want to support other sampling rates and data flows.

FreeSWITCH is designed from the ground up to be more flexible in its core, and uses as many existing libraries and tools as it can. Its development started in January 2006, and although there aren't any official releases at the time of writing this paper, the feature set is already complete—for a softswitch. Unfortunately there is only basic support for PSTN, a native module for Sangoma.

Another software package is Yate, started three years ago. It is written in C++, its source code is an order of magnitude smaller than Asterisk, and it has a cleaner architecture which grants much more flexibility. Yate can use the native WANPIPE interface to drive Sangoma hardware, delivering extremely high performance with high-density Sangoma cards.

6 Conclusion

Running telephony systems with Linux implementaions for the past three years has resulted in the following successful working setups:

1. Pure VoIP IVR and information service for call-center employees using Asterisk.

2. Software load testing of proprietary VoIP equipment using Asterisk.

3. VoIP exchange using Yate.

4. Softswitch with call-center and two E1 PSTN interfaces using Asterisk and Digium E1 equipment.

5. ISDN signalling proxy in Python, using Cronyx E1 equipment.

6. Hybrid PSTN/VoIP telephony network for Saratov State University—multiple gateways using Asterisk and (lately) OpenPBX plus OpenSER on Cronyx E1 equipment.

All implementations were based on i386 and x86_64 hardware platforms and Linux as the operating system kernel. Since these setups were put into operation, we have had no problems with the reliability, stability, or performance of the software we chose. This was a result of careful capacity planning, clustering, and $1+1$ reservations of critical components.

In this paper, we have provided reasons for why building a softswitch or PSTN-connected system from commodity hardware and open-source software may be desirable, and why Linux is a good platform for implementing voice services. However, there are some deficiencies in the current implementations, both in the kernel and in some of the opensource packages, that can potentially result in scalability issues. There are ways to avoid these issues or solve them completely. Our suggestions include improving the Zaptel framework or introducing a new, more efficient framework.

7 References

VOIP Wiki, `http://voip-info.org`

Nathan Willis. *Finding voice codecs for free software.* `http://software.newsforge.com/article.pl?sid=05/09/28/1646243`

Proceedings of the Linux Symposium

Volume One

June 27th–30th, 2007
Ottawa, Ontario
Canada