Вестник Сыктывкарского университета.
Серия 1: Математика. Механика. Информатика.
Выпуск 1 (21). 2016

УДК 004.057.4 004.7 004.056.5

КОДИРОВАНИЕ РЕЧЕВОЙ ИНФОРМАЦИИ В СИСТЕМАХ IP-ТЕЛЕФОНИИ

Л. С. Носов, В. В. Вечерский, В. С. Зудин, А. В. Можайкин

В данной статье рассмотрена защита речевой информации при ее передаче по системам IP-телефонии, поскольку данный канал потенциально подвержен вмешательству с целью нарушения конфиденциальности переговоров. Задача защиты речевой информации от перехвата актуальна как для обычных пользователей (в повседневных целях), так и для различных организаций, фирм или компаний во избежание перехвата коммерческих секретов конкурентами.

В работе предложен способ кодирования аудиоканала, создано собственное минималистичное программное обеспечение, позволяющее производить кодирование/декодирование речевой информации в частотной области.

Ключевые слова: IP-телефония, защита IP-телефонии, разборчивость речи.

Введение

Необходимость кодирования речевой информации возникла не так давно, но на сегодняшний день в связи с бурным развитием техники связи, особенно мобильной связи и IP-телефонии, решение этой проблемы имеет огромное значение при разработке систем связи [2].

Выбор средств и методов написания программы проводился на основе нескольких требований:

- программа должна являться утилитой, запускаемой из командной строки: разработка графического интерфейса для такой задачи

является нецелесообразным усложнением, к тому же «консольную» утилиту легче при необходимости встроить в программу на языке разработки сценариев;

- программа должна иметь возможность взаимодействия с другими программами звукозаписи и воспроизведения: предполагается, что основным применением программы может стать кодирование речи в системах IP-телефонии;

- программа должна быть способна работать «в реальном времени»: по той же причине.

Исходя из субъективных предпочтений разработку программы было решено проводить под семейство операционных систем *GNU/Linux*. Работа с командной строки является более естественной в этих операционных системах, чем, например, в ОС *Windows*.

Для выполнения второго требования необходимо было определить способ встраивания программы в существующий поток передачи звуковых данных какого-либо приложения. Рассматривались три возможных метода осуществить это:

- встраивание между сетевым интерфейсом и приложением;

- написание модуля для приложения с использованием собственного API этого приложения;

- встраивание между аппаратными средствами звукозаписи и воспроизведения (микрофон, колонки) и приложением (например, Skype).

Первый метод имеет крупные недостатки: круг приложений, с которыми может работать программа, ограничивается сетевыми приложениями, к тому же многие программы IP-телефонии используют специфичные, часто закрытые, протоколы передачи данных по сети. Второй метод ещё сильнее ограничивает возможности применения программы: она жёстко привязывается к тому приложению API, которое используется, а вообще говоря приложение может и не иметь собственного API. По этим причинам был выбран третий метод.

Первой частью реализации стало определение средств для обеспечения взаимодействия программы с аппаратными средствами звукозаписи и воспроизведения. Рассматривались три библиотеки, варианты которых были найдены для нескольких языков программирования [3,4]:

- библиотеки драйвера **OSS**;
- библиотеки драйвера **ALSA**;
- библиотеки звукового сервера **PulseAudio**.

OSS (Open Sound System) представляет собой унифицированный драйвер для звуковых карт и других звуковых устройств в UNIX-подобных операционных системах. Он является устаревшим, запрещает одновременное использование звукового устройства несколькими программами. В ходе решения задачи перенаправления звука оказалось, что использование низкоуровневых библиотек бессмысленно, и было решено отказаться от использования **OSS**.

ALSA (Advanced Linux Sound Architecture) – это более современный звуковой драйвер Linux-систем, призванный сменить **OSS** [4]. Обладает рядом преимуществ перед **OSS**, в том числе возможностью одновременного использования звукового устройства несколькими программами, но имеет заметно более сложный API. Большинство преимуществ **ALSA** не представляло никакого интереса, тем более в ходе решения задачи было решено, что использование **ALSA**-библиотеки стало слишком низкоуровневым для данного рода задачи [4].

PulseAudio представляет собой кросс-платформенный звуковой сервер. Он обладает огромным количеством преимуществ, наиболее интересными из которых в рамках данной задачи являются возможность перенаправления звуковых потоков и высокий уровень абстракции над звуковыми устройствами. Однако использование этого сервера сопровождается большими задержками, что, как оказалось в ходе решения задачи, не повлияло на работоспособность программы [3].

Следующим шагом было необходимо выбрать способ перенаправления звукового потока между программой и интересующим нас приложением IP-телефонии. Эта задача может решаться средствами звуковых серверов, среди которых были рассмотрены два возможных варианта [3, 5]:

- **JACK Audio Connection Kit**;
- **PulseAudio**.

Несмотря на то что сервер **JACK** предлагает большие возможности для перенаправления звуковых потоков, нами был выбран **PulseAudio**. Сделано это было по нескольким причинам:

- звуковой сервер **PulseAudio** является предустановленным во многих современных дистрибутивах Linux, в то время как **JACK** требуется устанавливать отдельно;

- тестирование программы предполагалось проводить на приложении Skype, Linux-версия которого требует обязательного наличия именно сервера **PulseAudio**.

В итоге было принято решение о использовании **API PulseAudio** [3]. Это решение позволило использовать преимущества этих библиотек, а также уменьшить число необходимых для работы программы компонентов.

1. Описание программы

Для перевода сигнала в частотную область и обратно используется быстрое преобразование Фурье (БПФ) и обратное быстрое преобразование Фурье (ОБПФ) [1]. Для обоих алгоритмов был реализован единый интерфейс, представленный функцией **fft()** (рис. 1). Это сделано для того, чтобы избежать дублирования кода, так как БПФ и ОБПФ имеют минимальные отличия, которые и учитывает функция **fft()**. Основной объём преобразований для обоих алгоритмов идентичен. Эти преобразования реализованы рекурсивно, в виде функции **fft_iteration()** (рис. 2).

БПФ удобно представляется с использованием комплексных чисел, поэтому также была написана минимальная реализация работы с ними.

Для выполнения кодирования речи был написан несложный алгоритм (рис. 3). В целях упрощения тестирования для ключа был придуман формат, легко воспринимаемый человеком, но при необходимости он без проблем может быть приведён к другому необходимому формату. Ключ представляет собой последовательность чисел в строковом представлении, разделённых двоеточием. Первое число ключа задаёт размер кадра для преобразования Фурье, а последующие числа определяют процесс кодирования. Все числа ключа, кроме размера кадра, в терминологии программы именуются «правилами». Так как БПФ работает только с массивами данных, размер которых равен степени двойки, то размер кадра устанавливается наибольшему числу, которое равно степени двойки и меньше числа, заданного в ключе. Собственно само кодирование происходит путём перемешивания значений в кадре, получаемом в результате выполнения преобразования Фурье над входными данными программы. Для этого кадр разбивается на блоки, размер которых задаётся очередным правилом из ключа, после чего порядок эле-

```
/*БПФ для data, size = n. Если inv == true тогда ОБПФ*/
void
fft(complex_t *data,int n,int inv){
  int i,j;
  int maskl,maskr;
  complex_t temp;
  for(i=0;i<n;i++){
    j=0;
    for(maskl=1,maskr=n>>1;maskl<n;maskl<<=1,maskr>>=1){
      if(maskl&i){
        j+=maskr;
      }
    }
    if(j>i){
      temp=data[i];
      data[i]=data[j];
      data[j]=temp;
    }
  }
  fft_iteration(data,n,inv);
  if (!inv){
    for(i=0;i<n;i++){
      data[i].re/=n;
      data[i].im/=n;
    }
  }
}
```

Рис. 1. Реализация общего интерфейса БПФ и ОБПФ

ментов в каждом блоке меняется на противоположный. Эти действия выполняются для каждого правила в ключе, благодаря чему можно достигнуть достаточной хаотичности элементов в кадре. После этого к сигналу применяется обратное быстрое преобразование Фурье (ОБПФ) и полученный результат отправляется на выход программы. Процесс декодирования выполняется так же, но правила читаются в обратном порядке.

Основная структура программы, представленная на рис. 4.

```
/*Рабочая фукнция для БПФ. Нельзя использовать напрямую!*/
void
fft_iteration(complex_t *data,int n,int inv){
  int m,i;
  complex_t temp,turn_rate;
  double step_arg;
  if(n==2){
    temp=data[0];
    data[0]=complex_add(data[0],data[1]);
    data[1]=complex_sub(temp,data[1]);
  }else{
    m=n/2;
    fft_iteration(data,m,inv);
    fft_iteration(data+m,m,inv);
    if(inv)
      step_arg=2*M_PI/n;
    else
      step_arg=-2*M_PI/n;
    for(i=0;i<m;i++){
      turn_rate=complex_from_exponent(1,step_arg*i);
      data[m+i]=complex_mult(data[m+i],turn_rate);
      temp=data[i];
      data[i]=complex_add(data[i],data[m+i]);
      data[m+i]=complex_sub(temp,data[m+i]);
    }
  }
}
```

Рис. 2. Рекурсивная реализация основы БПФ и ОБПФ

На этапе обработки опций программа узнаёт ключ и направление кодирования. Эта информация задаётся следующими опциями:

- **m** – направление (режим) кодирования. Может иметь два значения:
  - **code** – режим кодирования;
  - **decode** – режим декодирования.

```
void
code_frame(complex_t *frame,unsigned int frame_size,
unsigned int *rules,int n_rules){
  int i,j;
  int invert_size;
  for(i=0;i<n_rules;i++){
    for(j=0;j<frame_size/2;j+=rules[i]){
      invert_size=frame_size/2-j;
      if(invert_size>rules[i])
        invert_size=rules[i];
      invert_buf_complex(frame+j,invert_size);
    }
  }
  for(i=1,j=frame_size-1;i<j;i++,j--){
    frame[j].re=frame[i].re;
    frame[j].im=-1*frame[i].im;
  }
}
```

Рис. 3. Алгоритм кодирования речи

- **k** – ключ. Ключ передаётся как обычная строка, о формате которой мы уже говорили ранее.

Установка соединений с сервером **PulseAudio** происходит через **simple API** этого сервера [3]. Программа устанавливает соединение с устройствами ввода и вывода звукового сервера по умолчанию. С одной стороны, это позволяет не усложнять код программы, с другой — позволяет пользователю полностью контролировать взаимодействие нашей программы со средой средствами самого **PulseAudio**, для которого существует большое количество интерфейсов, в том числе и графических (рис. 5).

Далее программа считывает данные с устройства ввода, преобразует их, как было описано выше, и отправляет результат в устройство вывода. Эти действия повторяются до тех пор, пока программа не будет завершена пользователем или аварийно.

Благодаря использованию звукового сервера **PulseAudio**, программу удалось сделать полностью обособленной от процесса перенаправления потоков. Это позволяет сделать программу очень гибкой и встра-

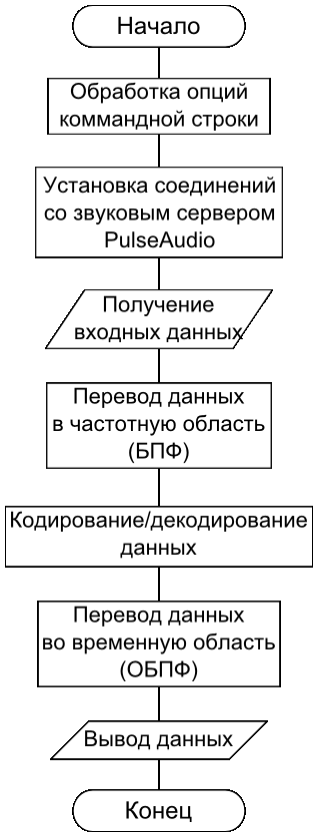

Рис. 4. Структура программы кодирования речи в частотной области

ивать её не только между приложением и аппаратными средствами, но и между двумя приложениями, причём делать это способами, которые пользователь сам сочтёт наиболее удобными. При тестировании программы, использовался общепринятый в **PulseAudio** способ перенаправления потоков через *null-sink* [3]. *Sink* (англ. сток) и *source* (англ. источник) – это обобщённые термины **PulseAudio**, применяемые к объектам, которые могут соответственно принимать звуковые данные и из которых можно читать звуковые данные. Простейшие примеры – это *sink*, связанный с колонками, или *source*, связанный с микрофоном. *Null-sunk* – это *sink*, данные в котором пропадают. Однако эти данные не исчезают бесследно. С каждым *sink* в **PulseAudio** связан *source*, который называется *monitor*. Этот *monitor* позволяет получить данные, которые были записаны в *sink*. Таким образом, *null-sink* может стать для двух приложений «точкой обмена» звуковыми данными, что и используется для перенаправления [3].

```
play_spec.format=PA_SAMPLE_S16LE;
play_spec.rate=8000; play_spec.channels=1;
rec_spec.format=PA_SAMPLE_S16LE;
rec_spec.rate=8000; rec_spec.channels=1;
play=pa_simple_new(NULL,argv[0],PA_STREAM_PLAYBACK,NULL,
"playback",&play_spec,NULL,NULL,&error);
if(play==NULL){
  fprintf(stderr,"%s: %s\n",argv[0],pa_strerror(error));
  retcode=1; goto finish;
}
rec=pa_simple_new(NULL,argv[0],PA_STREAM_PLAYBACK,NULL,
"record",&rec_spec,NULL,NULL,&error);
if(rec==NULL){
  fprintf(stderr,"%s: %s\n",argv[0],pa_strerror(error));
  retcode=1; goto finish;
}
for(;;){
  if(pa_simple_read(rec,buf,buf_size*2,&error)<0){
    fprintf(stderr,"%s: %s\n",argv[0],pa_strerror(error));
    retcode=1; goto finish;
  }
  /*Здесь выполяются действия по кодировнию*/
  if(pa_simple_write(play,buf,buf_size*2,&error)<0){
    fprintf(stderr,"%s: %s\n",argv[0],pa_strerror(error));
    retcode=1; goto finish;
  }
}
finish:
  if(play!=0)
    pa_simple_free(play);
  if(rec!=0)
    pa_simple_free(rec);
  /*Освобождаются прочие ресурсы*/
```

Рис. 5. Реализация взаимодействия со звуковым сервером PulseAudio

Например, при тестировании программы использовалась схема, изображённая на рис. 6.

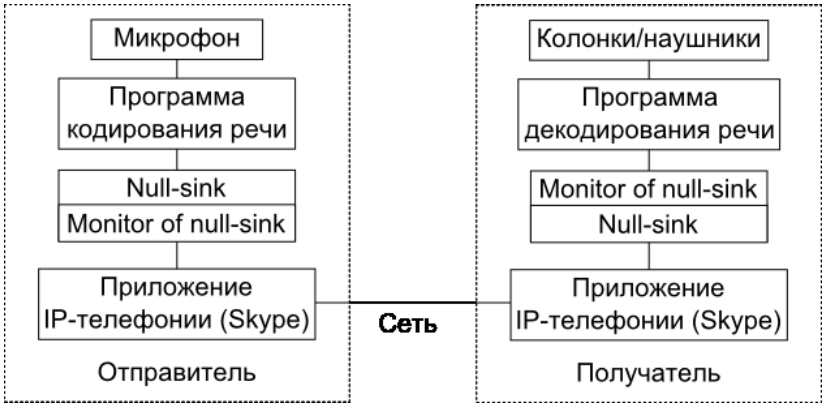

Рис. 6. Схема тестового стенда для программы кодирования речи

2. Оценка разборчивости речи, при кодировании в частотной области

В ходе тестирования программы было замечено, что в некоторых случаях закодированная речь остаётся распознаваемой.

Было сделано несколько предположений:

- при использовании ключа с одним правилом разборчивость речи повышается при уменьшении значения правила;
- при использовании ключа с двумя правилами, значения которых близки друг к другу, речь после кодирования остаётся понятной.

Для проверки этих предположений была проведена небольшая оценка разборчивости речевого сигнала на выходе кодера.

Для этого было записано звучание пятнадцати слов, после чего эти записи кодировались с использованием различных ключей и оценивалось число слов, которые возможно было разобрать на слух. Ещё при тестировании работоспособности программы стало понятно, что разборчивость речи зависит от размера кадра в меньшей степени, поэтому для него было выбрано постоянное значение 1024. Результаты проведённых экспериментов представлены в табл. 1. Относительная разборчивость здесь – это процентное соотношение числа разобранных слов к общему числу.

Из этой таблицы видно, что для ключей с одним правилом действительно свойственно повышение разборчивости при уменьшении значения ключа. Видимо это связано с тем, что при изменении порядка элементов в маленьких блоках общее распределение энергии в спектре сиг-

Таблица 1

Результаты экспериментов по оценке разборчивости закодированной речи

N	Ключ	Число разобранных слов	Отн. разборчивость %
1	1024:304	0	0
2	1024:43	1	6.7
3	1024:8	6	40
4	1024:765:640	15	100
5	1024:376:38	4	20
6	1024:53:980	3	26.7
7	1024:30:712:358	0	0
8	1024:484:167:947	1	6.7
9	1024:193:509:72	0	0

нала сильно не изменяется, поэтому некоторые слова остаются понятными.

Второе предположение также подтвердилось, хотя оно было очевидным само по себе, так как при близких значениях правил, второе изменение «отменяет» первое. Однако другой важный вывод заключается в том, что использование ключей с двумя правилами в принципе небезопасно: все проведённые эксперименты показали сравнительно высокий процент разборчивости. Видимо эту особенность следует отнести к недостаткам алгоритма.

Из-за проблем с ключами из двух правил были также проверены несколько значений ключа с тремя правилами. Взаимное влияние правил в этом случае уже принимает довольно сложную форму, поэтому делать предположения о качестве таких ключей сложно, однако проведённые измерения показали хорошие результаты.

Таким образом, была выявлена проблема созданного алгоритма: при его использовании возникает задача поиска «плохих» и «хороших» ключей. Судя по всему, похожая проблема возникает в принципе, когда кодирование осуществляется перестановками значений сигнала в частотной области. Если выполняемая перестановка недостаточно изменяет характер распределения энергии сигнала по частотам, то речь может оставаться понятной.

Заключение

В итоге была написана простая программа, реализующая кодирование речевого сигнала в частотной области. Полученная программа представляет собой гибкий инструмент, пригодный для тестирования различных методов кодирования речи. Сравнительно небольшими усилиями эта программа также может быть доведена до состояния полноценного компонента систем IP-телефонии.

Для проверки работоспособности программы был реализован и протестирован простой алгоритм кодирования речи, выполняемого путём перестановок значений сигнала в частотной области. Были выявлены недостатки этого алгоритма, а также недостатки кодирования перестановками в целом. При кодировании перестановками в частотной области возникает проблема «плохих» перестановок, которые недостаточно сильно изменяют распределение энергии по частотам. Это приводит к тому, что в некоторых ситуациях речь остаётся понятной, и секретная информация может быть перехвачена. Возможным решением может стать отход от техники перестановок в пользу, например, изменения значений амплитудного спектра.

В общем, в полученной программе удалось достаточно хорошо разделить процесс кодирования от остальных функций, поэтому её можно использовать для тестирования других алгоритмов в более объёмных исследованиях.

Список литературы

1. **James W. Cooley, John W. Tukey** An Algorithm for the Machine Calculation of Complex Fourier Series // *Mathematics of Computation, 1965. Pp. 297–301.*

2. **Юкио Сато.** Без паники! Цифровая обработка сигналов / пер. с яп. Т. Г. Селиной. М.: Додэка-XXI, 2010. 176 с.

3. PulseAudio Documentation // http://freedesktop.org: Software development management system. URL: http://freedesktop.org/software/pulseaudio/doxygen/ (дата обращения: 17.07.2016).

4. ALSA project - the C library reference. http://www.alsa-project.org: Advanced Linux Sound Architecture (ALSA) project homepage. URL: http://www.alsa-project.org/alsa-doc/alsa-lib/ (дата обращения: 17.07.2016).

5. JACK Audio Connection Kit. URL: http://www.jackaudio.org/ (дата обращения: 17.07.2016).

СГУ им. Питирима Сорокина *Поступила 18.07.2016*

Summary

Nosov L. S., Vecherskij V. V., Zudin V. S., Mozhajkin A. V. Encoding voice information in the IP-telephony

In this article, the voice data protection for its transmission over an IP-telephony systems is considered, since this channel is potentially exposed to interference in order to violate the confidentiality of negotiations. The challenge of protecting speech information from interception is relevant for ordinary users (daily use) and for various organizations, firms or companies in order to prevent the interception of commercial secrets by competitors.

In this paper we propose a method of encoding audio channels, create our own minimalist software that allows to encode / decode the speech information in the frequency domain.
Keywords: IP telephony, Protection of IP telephony, speech intelligibility.

References

1. **James W. Cooley, John W. Tukey** An Algorithm for the Machine Calculation of Complex Fourier Series // Mathematics of Computation, 1965. Pp. 297–301.

2. **Yukito Sato** Illustrated Introduction to Mechatronics. Introduction to Signal Management (Revised 2nd Edition). Tokyo: Ohmsha, 1999. 176 p.

3. PulseAudio Documentation // http://freedesktop.org: Software development management system. URL: http://freedesktop.org/software/pulseaudio/doxygen/ (date of the application: 17.07.2016).

4. ALSA project - the C library reference // http://www.alsa-project.org: Advanced Linux Sound Architecture (ALSA) project homepage. URL: http://www.alsa-project.org/alsa-doc/alsa-lib/ (date of the application: 17.07.2016).

5. JACK Audio Connection Kit // URL: http://www.jackaudio.org/ (date of the application: 17.07.2016).

Для цитирования: Носов Л. С., Вечерский В. В., Зудин В. С., Можайкин А. В. Кодирование речевой информации в системах IP-телефонии // Вестник Сыктывкарского университета. Сер. 1: Математика. Механика. Информатика. 2016. Вып. 1 (21). С.86–99.

For citation: Nosov L. S., Vecherskij V. V., Zudin V. S., Mozhajkin A. V. Encoding voice information in the IP-telephony // Bulletin of Syktyvkar University. Series 1: Mathematics. Mechanics. Informatics. 2016. №1 (21). Pp.86–99.