Документация по Speex
Version 1.2

Jean-Marc Valin

June 14, 2021

Contents

List of Tables

1. Введение в Speex

Кодек Speex (http://www.speex.org/) сушествует из-за необходимости в речевом кодеке у которого открытый исходный код и который свободен от патентных отчислений. Это необходимые условия для применимости в любых открытых программных продуктах. Важно заметить, что Speex для речи в то время как Vorbis для музыки/звуков. К сожалению в отличии от остальных речевых кодеков, Speex разработан не для мобильных телефонов, а для пакетных сетей и различных VoIP приложений. Сжатие в файлы, безусловно, также поддерживается.

Кодек Speex создан очень гибким и поддерживающим широкий диапазон качества речи и битрейта. Поддержка хорошего качества речи означает что Speex может кодировать широкополосную речь (частота дискретизации 16 кГц) вдобавок к узкополосной речи (телефонное качество, частота дискретизации 8 кГц).

Ориентированность на VoIP вместо мобильных телефонов означает что Speex стойкий к потере пакетов, но не к повреждению таковых. Это основывается на предположении что в VoIP пакеты либо приходят не измененными, либо не приходят вовсе. Так как Speex нацелен на широкий диапазон устройств, он обладает скромной и регулируемой сложностью и малым размером кода.

Все цели разработки приводят к выбору техники кодирования CELP. Одна из основных причин состоит в том, что давно доказано что CELP надежно работает и хорошо масштабируется при низких скоростях передачи данных (т.е. DoD CELP @ 4.8 kbps) и высоких скоростях (т.е. G.728 @ 16 kbps).

1.1. Помощь

Как и для большинства проектов с открытым исходным кодом существует множество способов получить помощь по Speex. Включая следующие:

- Данное руководство
- Остальная документация по на сайте Speex (http://www.speex.org/)
- Рассылка: обсуждаются все связанные со Speex темы на speex-dev@xiph.org (не только для разработчиков)
- IRC: Основной канал #speex на irc.freenode.net. Стоит заметить, что из-за разницы во времени, ожидание ответа может занять некоторое время, пожалуйста будьте терпеливыми.
- Отправьте email автору лично на jean-marc.valin@usherbrooke.ca только для личных вопросов, которые вы не хотите обсуждать публично.

Перед тем как просить помощи (через рассылку или IRC), важно прочесть данное руководство. Как правило неприлично спрашивать о тех разделах, которые детально и ясно описаны в руководстве. С другой стороны это правильно (мы даже одобряем это) просить объяснений по поводу того что не отражено в данном руководстве. Данное руководство не покрывает всего что связано со Speex, так что каждый может задавать вопросы, присылать комментарии, спрашивать о функционале, или сообщите нам о том как вы используете Speex.

Существуют еще некоторые дополнительные указания связанные с рассылкой. Перед отправкой отчета об ошибке, настоятельно рекомендуется (если возможно) сначала проверить, где проявляется данная ошибка, используя утилиты speecenc и speexdec (см. раздел 4). Ошибки связанные со сторонним кодом сложнее найти для обоих, и они очень часто вызваны ошибками не имеющими ничего общего со Speex.

1.2. Об этом документе

Данный документ структурирован следующим образом. Раздел 2 рассказывает про различные функции Speex и определяет многие основные термины, которые используются в данном руководстве. Раздел 4 документирует стандарт инструментов командной строки предоставленные в дистрибутиве Speex. Раздел

5 включает в себя детальные инструкции о программировании с использованием libspeex API. Раздел 7 содержит информацию относящуюся к Speex и стандартам.

Три последних раздела описывают алгоритмы используемые в Speex. Для понимания данных разделов требуются знание теоирии обработки сигналов, но для обычного использования Speex не нужно. Они предназначены для людей, которые хотят понять как работает Speex и/или желают провести некоторое исследование основанное на Speex. Раздел 8 содержит основные идеи CELP, в то время как разделы 10 и 11 относятся к самому кодеку.

2. Описание кодека

Этот раздел рассказывает о Speex и его функциях более детально.

2.1. Основные идеи

До введения в основные функции Speex, здесь представлены некоторые идеи касаемые кодирования звука которые помогают лучше понять остальную информацию. Хотя некоторые являются общими понятиями обработки голоса и звука, остальные специфичны для Speex.

Частота дискретизации

Частота дискретизации выраженная в герцах (Гц), это количество выборок сигнала которые берутся за один период. Для частоты дискретизации равной F_{s} кГц, наибольшая частота, которая может быть представлена равна F_{s}/2 кГц (F_{s}/2 также известна как частота Найквиста). Это фундаментальное свойство в обработке сигнала подтверждается теоремой Котельникова. Speex разработан для трех частот дискретизации: 8 кГц, 16 кГц, и 32 кГц. То есть, , широкополосный и ультра широкополосный соотвественно.

Битрейт

При кодировании речевого сигнала, скорость передачи данных (битрейт) определяется как количество битов в единицу времени, необходимое для кодирования речи. Он измеряется в битах в секунду (б/с), или как правило в килобитах в секунду. Также важно различать между *кило**битами** в секунду* (к**б**/с) и *кило**байтами** в секунду* (к**Б**с).

Качество (переменная)

Speex это кодек с потерями, это означает что он достигает сжатия ценой точности входного сигнала. В отличии от некоторых других речевых кодеков, здесь можно контролировать соотношение между качеством и объемом данных. Процесс кодирования Speex по большей части определячется параметром качества, который находится в диапаоне от 0 до 10. При постоянном битрейте (CBR), параметр качества целочисленный, в то время как для переменного битрейта (VBR), параметр с плавающей точкой.

Сложность (переменная)

В Speex можно варьировать сложность алгоритма кодировщика. Это выполняется через контроль осуществляемого поиска при помощи целого числа, варьирующегося в диапазоне от 1 до 10, данный параметр похож на опции -1 до -9 в утилитах сжатия gzip и bzip2. Для обычного использования уровень шума при сложности 1 на 1 или 2 дБ выше чем при сложности 10, но требования к процессору для сложности 10 приблизительно в пять раз выше чем для сложности 1. На практике наилучший компромисс находится между сложностью 2 и 4, хотя более высокие установки зачастую полезны при кодировании не речевых звуков, таких как тоны DTMF.

Переменный битрейт (VBR)

Переменный битрейт (VBR) позволяет кодеку менять битрейт динамически для адаптации к “сложности” кодируемого звука. Например в Speex звуки такие как гласные и высокоэнергетические импульсные помехи требуют большего битрейта для достижения хорошего качества, в то время как согласные (т.е. звуки с, ф) могут быть адекватно закодированы меньшим количеством бит. По этой причине с помощью VBR можно достичь меньшего битрейта для прежнего качества, или лучшего качества при прежнем битрейте. Несмотря на его преимущества, у VBR имеются два основных недостатка: во первых, при определении только качества, нет гарантий относительно среднего битрейта. Во-вторых некоторые приложения реального времени, такие

как VoIP, учитывают только максимальный битрейт, который должен быть достаточно низким для канала связи.

Средний битрейт (ABR)

Средний битрейт решает одну из проблем VBR, потому как он динамически регулирует качество VBR, чтобы поддерживать определенный целевой битрейт. Потому как качество/битрейт регулируется в реальном времени (цикл без обратной связи), глобальное качество будет немного ниже чем то, что получено путем кодирования в VBR с точной установкой качества для обеспечения целевого среднего битрейта.

Определение присутствия голосового сигнала (VAD)

Если включен, определитель присутствия голосового сигнала определяет звук, который будет кодироваться, им может быть речь или тишина/фоновый шум. VAD всегда неявно включается при кодировании в VBR, таким образом, эта опция полезна при работе без VBR. В данном случае Speex определяет периоды времени без голоса и кодирует их достаточны количеством бит для воспроизведения фонового шума. Это называется комфортной генерацией шума (CNG).

Прерывистая передача (DTX)

Прерывистая передача это дополнение к режимам VAD/VBR, которое позволяет полностью остановить передачу когда фоновый шум является стационарным. При работе с файлами, потому как мы не можем остановить запись в файл, используется только 5 бит для записи подобных кадров (что соответствует 250 бит/с).

Перцепционное усиление

Перцепционное усиление это часть декодировщика, которая при включении пытается уменьшить восприятие шума/искажений произведенных в процессе кодирования/декодирования. В большинстве случаев перцепционное усиление объективно удаляет звук от оригинального (например, сосредотачиваясь только на соотношении сигнал-шум), но в итоге звук лучше (субъективное улучшение).

Задержка и время работы алгоритма

Каждый речевой кодек вносит задержку в передачу. Для Speex эта задержка равна размеру кадра, плюс определенный запас, необходимый для обработки каждого кадра. В узкополосном режиме (8 кГц) запас составляет 10 мс, в широкополосном режиме (16 кГц) составляет 13.9 мс и в ультра широкополосном режиме (32 кГц) запас 15.9 мс, итоговая задержка алгоритма составляет 30 мс, 33.9 мс и 35.9 мс соответственно. Эти значения не учитывают время, которое необходимо процессору для кодирования и декодирования кадров.

2.2. Кодек

Основные характеристики Speex могут быть кратко изложен следующим образом:

- Свободное программное обеспечение с открытым исходным кодом, не имеет патентных ограничений и не требуетр лицензионных отчислений
- Совмещение узкополосного и широкополосного режимов с помощью замороченного битового потока
- Поддержка широкого диапазона битрейтов (от 2.15 кбит/с до 44 кбит/с)
- Режимы динамического переключение битрейта (AMR) и переменного битрейта (VBR)
- Определение пристуствия голосового сигнала (VAD, совмещено с VBR) и прерывистая передача (DTX)
- Варьируемая сложность алгоритма
- Встроенная широкополосная структура (масштабируемая частота дискретизации)
- Ультра-широкополосный режим с частотой дискретизации 32 кГц

- Intensity stereo encoding option
- Целочисленная реализавция

2.3. Препроцессор

Эта часть касается модуля препроцессора введенного в ветке 1.1.x. Препроцессор создан для обработки звука до запуска кодировщика. Препроцессор предоставляет три основных функции:

- подавление шума
- автоматическая регулировка усиления (AGC)
- определение присутствия голосового сигнала (VAD)

Шумоподавление может быть использовано для уменьшения количества фонового шума имеющегося во входном сигнале. Это обеспечивает более более высокое качество речи, при этом без разницы, кодируется ли сигнал при помощи Speex или нет. Однако при использовании сигнала с пониженным уровнем шума с кодеком существует дополнительная выгода. Речевые кодеки в целом (включая Speex) как правило обрабатывая плохой и зашумленный сигнал усиливают шум. Подавление шума значительно уменьшает этот эффект.

Автоматическая регулировка усиления (AGC) это функция которая имеет дело с тем, что записываемая громкость может варьироваться в широких пределах между различными установками. AGC предоставляет способ выравнивания сигнала относительно опорного значения громкости. Это полезно для VoIP потому как оно устраняет необходимость ручной регулировки микрофонного усиления. Второе преимущество в том что устанавливая усиление микрофона на умеренном (низком) уровне легче избежать клиппинга.

Определение отсутствия голосового сигнала предоставляемая препроцессором более функциональная, чем подобная функция предоставляемая кодеком.

2.4. Адаптивный буфер колебаний задержки

При передаче голоса (или любого другого содержимого в данном случае) через UDP или RTP, пакет может быть потерян, прийти с другой задержкой, или даже не прийти вовсе. Цель буфера колебаний задержки состоит в упорядочивании пакетов и их буферизация должна быть достаточно долгой (но не дольше чем необходимо) для того, чтобы их можно было отправить для декодирования.

2.5. Акустическое подавление эха

В любой системе громкой связи (Fig. 2.1) речь от удаленного источника воспроизводится локальными динамиками, распространяется по комнате и записывается микрофоном. Если звук, полученный микрофоном, отправляется непосредственно на удаленную конечную точку, тогда пользователь услышит эхо своего голоса. Акустический эхо-компенсатор создан для того, чтобы убрать акустическое эхо до того как оно отправится на удаленную конечную точку. Для тех кто много беспокоится о задержках следует заметить что в отличии от кодека Speex, устройство передискретизации и препроцессор, и данный акустический эхо-компенсатор не вносит какой либо заметной задержки.

2.6. Устройство передискретизации

В некоторых случаях может быть полезным конвертирование звука из одной частоты дискретизации в другую. Существует несколько причин для этого. Это может использоваться для смешивания потоков у которых разные частоты дискретизации, для поддержки частот дискретизации, не поддерживаемых звуковой картой, для перекодировки, и т.д. Поэтому теперь устройство передискретизации это часть проекта Speex. Данное устройство может быть использовано для конвертирования между любыми двумя произвольными частотами дискретизации (соотношение должно быть рациональным числом), а также существует контроль над соотношением сложности и качества. Помните что устройство передискретизации вводит некоторую задержку в звуковой поток, чей размер зависит от установки сложности устройства передискретизации. Обратитесь к документации на API для того чтобы узнать как получить точные значения задержек.

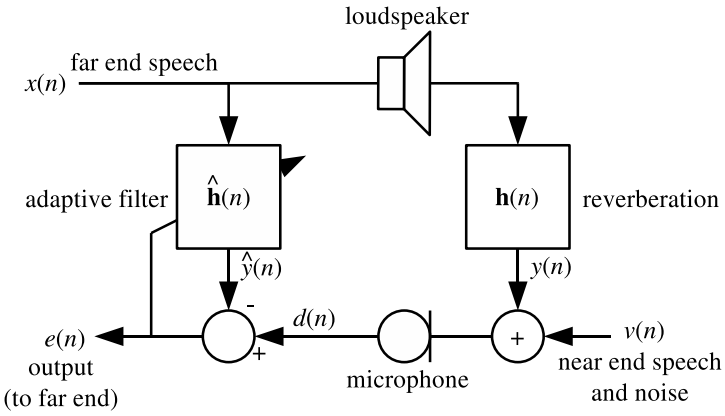


Figure 2.1.: Акустическая модель эха

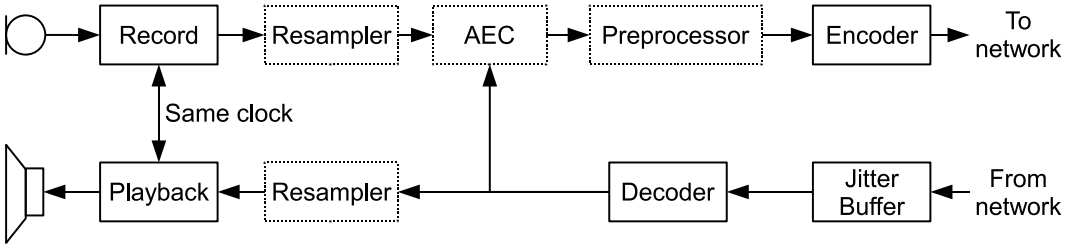


Figure 2.2.: Интеграция всех компонентов в VoIP клиенте.

2.7. Интеграция

Знание о том как использовать каждый из компонентов не настолько полезно, если мы не знаем где их использовать. Рисунок 2.2 показывает где каждый из компонентов может быть использован в типичном VoIP клиенте. Компоненты из пунктирных линий опциональны, хотя они могут быть полезны при некоторых обстоятельствах. Существует несколько моментов, о которых следует здесь упомянуть. AEC должен быть размещен настолько близко насколько это возможно к воспроизводящей цепи и цепи захвата. Только ресемплер может быть расположен ближе. Также очень важно использовать одинаковое тактирование для захвата с микрофона и воспроизведения на динамиках/наушниках.

3. Компиляция и портирование

Компилирование Speex под UNIX/Linux или множестве других платформ поддерживающих autoconf (например Win32/cygwin) с легкостью осуществляется путем написания:

```
% ./configure [options]
% make
% make install
```

Опции, поддерживаемые конфигурационным сценарием:

–prefix=<path> Определяет путь установки Speex (например /usr)

–enable-shared/–disable-shared Компилировать или не компилировать разделяемые библиотеки

–enable-static/–disable-static Компилировать или не компилировать статические библиотеки

–disable-wideband Отключить широкополосную часть Speex (обычно для экономии памяти)

–enable-valgrind Включить дополнительные hits для valgrind для отладки (не используется по умолчанию)

–enable-sse Включить использование инструкций SSE (только для x86/float)

–enable-fixed-point Компилировать для процессора не умеющего в плавающую точку (FPU)

–enable-arm4-asm Включить ассемблерную оптимизацию для архитектуры ARMv4 (только для gcc)

–enable-arm5e-asm Включить ассемблерную оптимизацию для архитектуры ARMv5E (только для gcc)

–enable-fixed-point-debug Используется только для отладки целочисленного кода (очень медленная)

–enable-ti-c55x Включить поддержку семейства TI C5x

–enable-blackfin-asm Включить ассемблерную оптимизацию для архитектуры Blackfin DSP (только для gcc)

3.1. Платформы

Speex известен как компилируемый и работающий на множестве платформ и архитектур, для чисел с плавающей точкой и фиксированной точкой. В целом, любая архитектура, которая изначально может осуществлять умножение для двух знаковых 16 разрядных чисел (с 32-х битным результатом) и работать на достаточной тактовой частоте способна запустить Speex. Архитектуры на которых **известна** работоспособность Speex (может работать на множестве других):

- x86 & x86-64
- Power
- SPARC
- ARM
- Blackfin
- Coldfire (семейство 68k)
- TI C54xx & C55xx
- TI C6xxx

- TriMedia (экспериментально)

Операционные системы на которых на которых замечена работоспособность Speex включают в себя (может работать на множестве других):

- Linux
- μClinux
- MacOS X
- BSD
- Остальные варианты UNIX/POSIX
- Symbian

Директория с исходным кодом включает в себя дополнительную информацию о компилировании на определенных архитектурах и операционных системах в файлах README.xxx.

3.2. Портирование и оптимизация

Здесь описаны некоторые моменты, рассматриваемые при портировании или оптимизации Speex для новой или уже существующей платформы.

3.2.1. Оптимизация под архитектуру

Единственный фактор который значительно влияет на загрузку процессора кодеком Speex состоит в том, скомпилирован ли он для вычислений с плавающей точкой или с фиксированной. Если ваш ЦПУ/ЦСП не имеет модуля обработки чисел с плавающей точкой, тогда скомпилированный с фиксированной точкой кодек будет на порядки быстрее. На x86 архитектуре, вариант с плавающей точкой в основном быстрее, но не всегда. Для того, чтобы скомпилировать целочисленный вариант Speex, необходимо передать параметр –fixed-point конфигурационному сценарию или определить макрос FIXED_POINT для компилятора. По состоянию на 1.2beta3, теперь появилась возможность выключить API совместимости с плавающей точкой, это означает, что программа будет подключаться без библиотеки эмуляции плавающей точки. Для того чтобы это сделать, запустите конфигурацию с параметром –disable-float-api или определите макрос DISABLE_FLOAT_API. До тех пор пока функция VBR не портирована для целочисленного варианта, вам также необходимо запускать конфигурацию с параметром –disable-vbr или определить макрос DISABLE_VBR.

Остальные важные моменты, которые необходимо проверить на некоторых архитектурах сигнальных процессоров:

- Убедитесь в том что кэш установлен в режим с отложенной записью
- Если чип имеет статическое ОЗУ вместо кэша, убедитесь сколько кода и данных содержится в статическом ОЗУ, а не в ОЗУ

Если вы собираетесь писать на ассемблере, следующие функции **обычно** необходимо рассматривать для оптимизации в первую очередь:

- filter_mem16()
- iir_mem16()
- vq_nbest()
- pitch_xcorr()
- interp_pitch()

Функции фильтров filter_mem16() и iir_mem16() реализованы в виде транспонированной прямой формы второго типа (DF2T). Хотя для архитектур, основанных умножении с накоплением (MAC), DF2T требует частой перезагрузки аккумулятора, что может сделать программу очень медленной. Для данных архитектур (т.е. Blackfin и Coldfire), лучший подход состоит в применении первой прямой формы (DF1), который проще выражается в элементах MAC. **Важно убедиться в том, реализация DF1 ведет себя также как оригинальная реализация DF2T при записи значений в память.** Это необходимо потому что фильтр меняющийся во времени и должен вычислить точно такое же значение (без учета ошибки округления) на кодировщике и декодировщике.

3.2.2. Оптимизация памяти

Оптимизация памяти это то, что большей частью должно учитываться для малых встраиваемых платформ. Для ПК Speex уже настолько мал что не стоит делать вещей предлагаемых здесь. Существует несколько способов уменьшить использование памяти в Speex, как для объема кода, так и для объема данных. Для оптимизации объема кода трюк состоит в удалении ненужных вам функций. Некоторые функции могут быть с легкостью удалены если **вы не нуждаетесь в них**:

- Поддержка широкополосного режима (–disable-wideband)
- Поддержка стерео (удаление stereo.c)
- поддержка VBR (–disable-vbr или DISABLE_VBR)
- Статические кодовые таблицы которые не нужны, для битрейтов, которые вы не используете (*_table.c files)

У Speex также имеется несколько методов для размещения временных массивов. При использовании компилятора, который поддерживает C99 (по состоянию на 2007 год, компиляторы Microsoft не поддерживают, но gcc поддерживает), лучше всего будет определить VAR_ARRAYS. Это разрешит использование функции массивов переменной длины C99. Следующим полезным шагом является определение USE_ALLOCA, в этом случае Speex будет использовать alloca() для выделения памяти для временных массивов. Заметим что во множестве систем, alloca() работает с ошибками и может не работать вовсе. Если VAR_ARRAYS и USE_ALLOCA не определены, тогда Speex возвращается к выделению больших "временных областей" и внутреннему разделению выделенной памяти. Основной недостаток данного решения состоит в том что оно требует много ресурсов. Необходимо выделить достаточно памяти для худшего сценария (худший битрейт, наивысший параметр сложности, ...) и по умолчанию память не разделяется между множеством состояний кодировщика/декодировщика. Тем не менее, если ручное выделение памяти является единственным оставшимся вариантом, существует несколько моментов, которые могут быть улучены. Переопределяя вызов функции the speex_alloc_scratch() в os_support.h, зачастую можно возвращать одну и ту же память для всех состояний[1]. Вдобавок к этому, переопределением NB_ENC_STACK и NB_DEC_STACK (или подобных для широкополосного режима), возможно выделить память только для того сценария, который известен ранее. В этом случае важно измерить количество памяти необходимого для определенной частоты выборок, битрейта и уровня сложности который используется.

[1] In this case, one must be careful with threads

4. Консольные кодировщик и декодировщик

Базовый дистрибутив Speex включает в себя консольный кодировщик (speexenc) и декодировщик (speexdec). Эти инструменты создают и читают файлы Speex, упакованные в контейнере Ogg. Хотя возможно инкапсулировать Speex в любой контейнер, Ogg является рекомендуемым контейнером для подобных файлов. Данный раздел рассказывает о том, как использовать инструменты командной строки для файлов Speex в контейнере Ogg.

4.1. *speexenc*

Утилита speexenc используется для создания файлов Speex из raw PCM или wave файлов. Функцией можно воспользоваться путем вызова:

```
speexenc [параметры] входной_файл выходной_файл
```

Значение '-' для входного_файла или выходного_файла соответствует stdin и stdout соответственно. Доступны следующие опции:

–narrowband (-n) Включает обработку входного потока как узкополосного (8 кГц). Эта опция установлена по умолчанию.

–wideband (-w) Включает обработку входного потока как широкополосного (16 кГц).

–ultra-wideband (-u) Включает обработку входного потока как ультра широкополосного (8 кГц).

–quality n Устанавливает качество кодирования (0-10), по умолчанию 8

–bitrate n Кодируемый битрейт (использует bit-rate n или ниже)

–vbr Включить VBR (переменный битрейт), по умолчанию отключен

–abr n Включить ABR (средний битрейт) равный n кбит/с, по умолчанию отключен

–vad Включить VAD (определение присутствия голосового сигнала), по умолчанию отключен

–dtx Включить DTX (прерывистая передача), по умолчанию отключена

–nframes n Упаковывать n кадров в каждый Ogg пакет (это экономит память на низких битрейтах)

–comp n Установить соотношение скорости/качества. Более высокое значение n замедляет кодирование (по умолчанию 3)

-V Подробный режим, показывает битрейт, используемый в данный момент

–help (-h) Вывести на экран помощь

–version (-v) Показать информацию о версии ПО

Комментарии Speex

–comment Добавить данную строку как дополнительный комментарий. Может быть использовано несколько раз.

–author Автор трека.

–title Заголовок трека.

Raw input options

–rate n Sampling rate for raw input

–stereo Consider raw input as stereo

–le Raw input is little-endian

–be Raw input is big-endian

–8bit Raw input is 8-bit unsigned

–16bit Raw input is 16-bit signed

4.2. *speexdec*

Утилита speexdec используется для декодирования файлов Speex и может быть использована путем вызова:

```
speexdec [опции] speex_файл [выходной_файл]
```

Значение '-' для speex_файл или выходной_файл соответствует stdin и stdout. Также, когда не определен выходной файл, файл воспроизводится через звуковую карту. Доступны следующие опции:

–enh Включить пост-фильтр (по умолчанию)

–no-enh Выключить пост-фильтрацию

–force-nb Установить декодирование в узкополосном режиме

–force-wb Установить декодирование в широкополосном режиме

–force-uwb Установить декодирование в ультра широкополосном режиме

–mono Установить декодирование в режиме моно

–stereo Установить декодирование в режиме стерео

–rate n Установить декодирование на частоте дискретизации n Гц

–packet-loss n Имитировать n % случайной потери пакетов

-V Подробный режим, показывает битрейт, используемый в данный момент

–help (-h) Вывести на экран помощь

–version (-v) Показать информацию о версии ПО

5. Использование API кодека Speex (*libspeex*)

Библиотека libspeex содержит все функции для кодирования и декодирования речи с использованием кодека Speex. При подключении в операционных системах UNIX необходимо добавить -lspeex -lm в командную строку компилятора. Одна важная вещь которую необходимо знать, это то что вызовы функций libspeex могут быть сделаны повторно, **но при этом данная библиотека не ориентирована на многопоточное исполнение.** Это означает, что нужно быть осторожным при вызове из нескольких потоков, **но вызовы использующие одинаковое состояние из нескольких потоков должны быть защищены мьютексами.** Примеры программ также могут быть найдены в приложении A и полная документация по API включена в раздел Documentation на сайте Speex (http://www.speex.org/).

5.1. Encoding

Для того, чтобы кодировать речь с использованием Speex, во-первых необходимо:

```
#include <speex/speex.h>
```

Далее в программе должна быть объявлена структура упаковщика битов Speex вместе с переменной состояния кодировщика Speex:

```
SpeexBits bits;
void *enc_state;
```

Во вторых, для инициализации:

```
speex_bits_init(&bits);
enc_state = speex_encoder_init(&speex_nb_mode);
```

Для широкополосного режима, *speex_nb_mode* должна быть заменена *speex_wb_mode*. В большинстве случаев, вам необходимо знать размер кадра при используемой вами частоте дискретизации. Вы можете получить это значение через переменную *frame_size* (выраженную в **выборках**, а не в байтах) с помощью:

```
speex_encoder_ctl(enc_state,SPEEX_GET_FRAME_SIZE,&frame_size);
```

На практике, *frame_size* должен соответствовать 20 миллисекундам при использовании частоты дискретизации в 8, 16, или 32 кГц. Существует множество параметров, которые должны быть установлены для кодировщика Speex, но наиболее полезный из них это параметр качества, который контролирует соотношение между качеством и битрейтом. Он устанавливается с помощью:

```
speex_encoder_ctl(enc_state,SPEEX_SET_QUALITY,&quality);
```

где quality это целое число между 0 и 10 (включительно). Соотношение между качеством и битрейтом показано на рисунке 10.1 для узкополосного режима.

После того как инициализация произведена, для каждого входящего кадра:

```
speex_bits_reset(&bits);
speex_encode_int(enc_state, input_frame, &bits);
nbBytes = speex_bits_write(&bits, byte_ptr, MAX_NB_BYTES);
```

где *input_frame* это *(short *)* указатель на начало речевого кадра, *byte_ptr* это *(char *)* это указатель на начало закодированного кадра, MAX_NB_BYTES это максимальное число байт, которые могут быть записаны в *byte_ptr*, не вызывая переполнения стека, и *nbBytes* это число байт, которое записано в *byte_ptr* (размер закодированного кадра в байтах). До вызова *speex_bits_write*, можно определить количество байт, которые будут записаны, с помощью вызова `speex_bits_nbytes(&bits)`, которая возвращает количество байт.

Возможно использование функции *speex_encode()*, которая использует *(float *)* для звука. Хотя это может сделать порт на платформу без модуля обработки операций с плавающей точкой (например ARM) более

медленным. Внутренне *speex_ encode()* и *speex_ encode_ int()* работают одинаково. Использование кодировщиком целочисленного варианта или варианта с плавающей точкой определяется только флагами компилятора, а не на уровне API.

После того, как вы выполнили кодирование, освободите ресурсы через:

```
speex_bits_destroy(&bits);
speex_encoder_destroy(enc_state);
```

На этом все.

5.2. Декодирование

Для декодирования речи с использованием Speex, сначала вам нужно:

```
#include <speex/speex.h>
```

Также необходимо объявить структуру

```
SpeexBits bits;
```

и переменную состояния декодера

```
void *dec_state;
```

Они инициализируются с помощью:

```
speex_bits_init(&bits);
dec_state = speex_decoder_init(&speex_nb_mode);
```

Для широкополосного режима, *speex_ nb_ mode* должна быть заменена *speex_ wb_ mode*. В большинстве случаев, вам необходимо знать размер кадра при используемой вами частоте дискретизации. Вы можете получить это значение через переменную *frame_ size* (выраженную в **выборках**, а не байтах) с помощью:

```
speex_decoder_ctl(dec_state, SPEEX_GET_FRAME_SIZE, &frame_size);
```

Также существует параметр, который должен быть установлен для декодировщика: используется ли перцепционный усилитель или нет. Он должен быть установлен с помощью:

```
speex_decoder_ctl(dec_state, SPEEX_SET_ENH, &enh);
```

где *enh* целое значение равное 0 для выключения усилителя и 1 для включения. Для 1.2-beta1, по умолчанию он включен.

Опять, после того как инициализация декодера завершена, для каждого входящего кадра:

```
speex_bits_read_from(&bits, input_bytes, nbBytes);
speex_decode_int(dec_state, &bits, output_frame);
```

где input_bytes это *(char *)* содержащий потоковые данные для одного кадра, *nbBytes* размер (в байтах) этого потока, и *output_ frame* это *(short *)* которое указывает на область, в которой будут записаны декодированные речевые кадры. Значение NULL в качестве второго аргумента показывает что у нас нету данных для текущего кадра. Когда кадр теряется, декодер Speex пытается “угадать” правильный сигнал.

Как и для кодировщика, функция *speex_ decode()* также может быть использована с *(float *)*, как выходными данными для звука. После того как вы завершили декодирование, освободите все ресурсы с помощью:

```
speex_bits_destroy(&bits);
speex_decoder_destroy(dec_state);
```

5.3. Опции кодека (speex_*_ctl)

Не следует множить сущее без необходимости – Уильям Оккам.

Потому что наличие опции для этого не означает что ты должен её включать - я.

Кодировщик и декодировщик Speex поддерживают множество опций и запросов, которые могут быть доступны через функции *speex_encoder_ctl* и *speex_decoder_ctl*. Эти функции схожи с системными вызовами *ioctl* и их прототипы имеют следующий вид:

```
void speex_encoder_ctl(void *encoder, int request, void *ptr);
void speex_decoder_ctl(void *encoder, int request, void *ptr);
```

Несмотря на наличие этих функций, значения по умолчанию обычно подходят для большинства приложений и опциональные настройки **могут быть использованы только тогда, когда вы понимаете что они означают, и знаете для чего они вам нужны.** Основная ошибка состоит в попытке установить множество ненужных настроек.

Здесь представлен список значений, применяемых для запросов. Некоторые из них применяются только для кодировщика или декодировщика. Потому как последний аргумент имеет тип void *, функции _ctl() не типобезопасны, **таким образом вы должны их использовать с осторожностью**. Тип spx_int32_t это тип int32_t в C99.

SPEEX_SET_ENH‡ Установить перцепционный усилитель для включения (1) или (0) для того чтобы выключить (spx_int32_t, по умолчанию включен)

SPEEX_GET_ENH‡ Получить статус перцепционного усилителя (spx_int32_t)

SPEEX_GET_FRAME_SIZE Получить количество выборок на кадр текущем режиме (spx_int32_t)

SPEEX_SET_QUALITY† Установить качество кодируемой речи (spx_int32_t от 0 до 10, по умолчанию 8)

SPEEX_GET_QUALITY† Получить текущее значение качества кодируемой речи (spx_int32_t от 0 до 10)

SPEEX_SET_MODE† Установить режим под номером, как определено в RTP спецификации (spx_int32_t)

SPEEX_GET_MODE† Получить режим под номером, как определено в RTP спецификации (spx_int32_t)

SPEEX_SET_VBR† Установить переменный битрейт (VBR) (1) чтобы включить, (0) чтобы выключить (spx_int32_t, по умолчанию выключен)

SPEEX_GET_VBR† Получить статус переменного битрейта (VBR) (spx_int32_t)

SPEEX_SET_VBR_QUALITY† Установить качество кодируемой речи при включенном VBR (float 0.0 до 10.0, по умолчанию 8.0)

SPEEX_GET_VBR_QUALITY† Получить значение качества кодируемой речи при включенном VBR (float от 0 до 10)

SPEEX_SET_COMPLEXITY† Установить ресурсы ЦПУ, разрешенные для кодировщика (spx_int32_t from 1 to 10, default is 2)

SPEEX_GET_COMPLEXITY† Получить ресурсы ЦПУ, разрешенные для кодировщика (spx_int32_t от 1 до 10)

SPEEX_SET_BITRATE† Установить битрейт, использует ближайшее значение, не превышающее параметр (spx_int32_t в битах в секунду)

SPEEX_GET_BITRATE Получить текущий используемый битрейт (spx_int32_t в битах в секунду)

SPEEX_SET_SAMPLING_RATE Установить частоту дискретизации (spx_int32_t в Гц)

SPEEX_GET_SAMPLING_RATE Получить частоту дискретизации (spx_int32_t в Гц)

SPEEX_RESET_STATE Сбросить состояние кодировщика/декодировщика, стирает всю память (без аргумента)

SPEEX_SET_VAD† Установить значение определения присутствия голосового сигнала(VAD), (1) чтобы включить, (0) чтобы выключить (spx_int32_t, по умолчанию выключен)

SPEEX_GET_VAD† Получить статус определения присутствия голосового сигнала (VAD) (spx_int32_t)

SPEEX_SET_DTX† Установить прерывистую передачу (DTX), (1) чтобы включить, (0) чтобы выключить (spx_int32_t, по умолчанию выключен)

SPEEX_GET_DTX† Получить статус прерывистой передачи (DTX) (spx_int32_t)

SPEEX_SET_ABR† Установить значение среднего битрейта (ABR) в n бит в секунду (spx_int32_t в битах в секунду)

SPEEX_GET_ABR† Получить значение среднего битрейта при (ABR) (spx_int32_t в битах в секунду)

SPEEX_SET_PLC_TUNING† Сказать кодировщику оптимизировать кодирование для определенного процента потери пакетов (spx_int32_t в процентах)

SPEEX_GET_PLC_TUNING† Получить текущее значение PLC (spx_int32_t в процентах)

SPEEX_GET_LOOKAHEAD Возвращает lookahead используемый Speex отдельно для кодировщика и декодировщика. Сумма значений lookahead кодировщика и декодировщика это общее значение lookahead.

SPEEX_SET_VBR_MAX_BITRATE† Установить максимальный битрейт допустимый при использовании VBR (spx_int32_t in bits per second)

SPEEX_GET_VBR_MAX_BITRATE† Получить максимальный битрейт допустимый при использовании VBR (spx_int32_t в битах в секунду)

SPEEX_SET_HIGHPASS Установить фильтр верхних частот в (1) или выключить (0) (spx_int32_t, по умолчанию включен)

SPEEX_GET_HIGHPASS Получить текущий статус фильтра верхних частот (spx_int32_t)

† применяются только для кодировщика

‡ применяются только для декодировщика

5.4. Запросы режима

Режимы Speex имеют систему запросов похожую на speex_encoder_ctl и speex_decoder_ctl. Так как режимы доступны только для чтения, возможно только получать информацию о конкретном режиме. Функция осуществляющая это представлена ниже:

```
void speex_mode_query(SpeexMode *mode, int request, void *ptr);
```

Допустимые значения для запросов следующие (если не указано иное, значения возвращаются через *ptr*):

SPEEX_MODE_FRAME_SIZE Получить размер кадра (в выборках) для режима

SPEEX_SUBMODE_BITRATE Получить битрейт для подрежима определенного через *ptr* (целое в бит/с).

5.5. Формирование пакетов и внутридиапазонная связь

Иногда желательно разместить больше одного кадра на каждый пакет (или другую базовую единицу хранения данных). Правильный способ сделать это, это вызвать speex_encode N раз до записи потока с помощью speex_bits_write. В случаях где количество кадров не определено внеполосным механизмом, возможно включить код разделителя. Данный разделитель состоит из кода 15 (в десятичной системе счисления) закодированного пятью битами, как показано в таблице 10.1. Заметим что на момент версии 1.0.2, вызов speex_bits_write автоматически вставляет разделитель к моменту заполнения последнего байта. Он не включает в себя какого либо заголовка, поэтому убедитесь, что Speex всегда определяет момент, когда нет больше кадров в пакете.

Также возможно отправить внутридиапазонные "сообщения" другой стороне. Все данные сообщения кодируются как псевдо кадры режима 14, который содержит 4-х битовый код типа сообщения с последующим сообщением. Таблица 5.1 содержит список допустимых кодов, их значение и размер сообщения, следующего после. Большинство из этих сообщений это запросы, которые отправляются кодировщику или декодировщику на другой конец, который может соблюдать или игнорировать их. По умолчанию все внутридиапазонные сообщения игнорируются.

Код	Размер (бит)	Содержимое
0	1	Запрашивает у декодировщика установить перцепционное усиление выключенным (0) или включе
1	1	Запрашивает (если 1) у кодировщика быть менее “агрессивным” к большим потерям пакет
2	4	Запрашивает у кодировщика переключится в режим N
3	4	Запрашивает у кодировщика переключится в режим N для нижнего диапазона
4	4	Запрашивает у кодировщика переключится в режим N для верхнего диапазона
5	4	Запрашивает у кодировщика переключится на качество N для VBR
6	4	Запросить подтверждение (0=нет, 1=все, 2=только для внутридиапазонных данных)
7	4	Запрашивает у кодировщика установить CBR (0), VAD(1), DTX(3), VBR(5), VBR+DTX(
8	8	Передать (восьмиразрядное) значение в другой конец
9	8	Информация об интенсивности стерео
10	16	Объявить максимально допустимый битрейт (N в байтах в секунду)
11	16	зарезервировано
12	32	Подтверждение приема пакета N
13	32	зарезервировано
14	64	зарезервировано
15	64	зарезервировано

Table 5.1.: Коды внутридиапазонной связи

Наконец, приложения могут определять собственные внутридиапазонные сообщения, используя режим 13. Размер сообщения в байтах кодируется пятью битами, так что декодировщик может пропустить их, если не знает как их интерпретировать.

6. API обработки звука (*libspeexdsp*)

На момент версии 1.2beta3, части пакета Speex, не относящиеся к кодеку, теперь расположены в отдельной библиотеке названной *libspeexdsp*. Данная библиотека включает в себя препроцессор, акустическое подавление эха, буфер колебаний задержки, и устройство передискретизации. В среде UNIX она может быть подключена к программе путем добавления *-lspeexdsp -lm* в командной строке компилятора. Также как и в случае с libspeex, **вызовы libspeexdsp реентрабельны, но не потокобезопасны.** Это означает что нужно быть осторожным при вызове из нескольких потоков, **но вызовы, использующие одинаковое состояние из нескольких потоков, должны быть защищены мьютексами.**

6.1. Препроцессор

Для того, чтобы использовать препроцессор Speex, вам необходимо:

```
#include <speex/speex_preprocess.h>
```

После чего, состояние препроцессора может быть установлено через:

```
SpeexPreprocessState *preprocess_state = speex_preprocess_state_init(frame_size, sampling_rat
```

и рекомендуется использовать те значения `frame_size` которые использовались для кодировщика (20 *мс*).

Для каждого входящего кадра необходимо вызывать:

```
speex_preprocess_run(preprocess_state, audio_frame);
```

где `audio_frame` используется и для входного и для выходного потока. В случаях, когда нет полезного звука на выходе в текущем кадре, можно использовать функцию:

```
speex_preprocess_estimate_update(preprocess_state, audio_frame);
```

Данный вызов обновит все внутренние переменные состояния препроцессора без вычисления выходного звука, таким образом экономя ресурсы процессора.

Поведение препроцессора может быть изменено с помощью:

```
speex_preprocess_ctl(preprocess_state, request, ptr);
```

которая используется так же как её эквивалент для кодировщика и декодировщика. Опции описаны в разделе 6.1.1.

Состояние препроцессора может быть уничтожено с помощью:

```
speex_preprocess_state_destroy(preprocess_state);
```

6.1.1. Опции препроцессора

Как и в случае с кодеком, препроцессор также имеет опции, которые контролируются при помощи ioctl() подобных вызовов. Доступны следующие опции:

SPEEX_PREPROCESS_SET_DENOISE Включает шумоподавление, (1) чтобы включить, (0) чтобы выключить (spx_int32_t)

SPEEX_PREPROCESS_GET_DENOISE Возвращает состояние устройства шумоподавления (spx_int32_t)

SPEEX_PREPROCESS_SET_AGC Включает автоматическую регулировку усиления (AGC), (1) чтобы включить, (0) чтобы выключить (spx_int32_t)

SPEEX_PREPROCESS_GET_AGC Возвращает состояние AGC (spx_int32_t)

SPEEX_PREPROCESS_SET_VAD Включает детектор активности звука (VAD), (1) чтобы включить, (0) чтобы выключить (spx_int32_t)

SPEEX_PREPROCESS_GET_VAD Возвращает состояние VAD (spx_int32_t)

SPEEX_PREPROCESS_SET_AGC_LEVEL

SPEEX_PREPROCESS_GET_AGC_LEVEL

SPEEX_PREPROCESS_SET_DEREVERB Включает удаление реверберации, (1) чтобы включить, (0) чтобы выключить (spx_int32_t)

SPEEX_PREPROCESS_GET_DEREVERB Возвращает состояние удалятора реверберации (spx_int32_t)

SPEEX_PREPROCESS_SET_DEREVERB_LEVEL На данный момент не работает, не используйте

SPEEX_PREPROCESS_GET_DEREVERB_LEVEL На данный момент не работает, не используйте

SPEEX_PREPROCESS_SET_DEREVERB_DECAY На данный момент не работает, не используйте

SPEEX_PREPROCESS_GET_DEREVERB_DECAY На данный момент не работает, не используйте

SPEEX_PREPROCESS_SET_PROB_START

SPEEX_PREPROCESS_GET_PROB_START

SPEEX_PREPROCESS_SET_PROB_CONTINUE

SPEEX_PREPROCESS_GET_PROB_CONTINUE

SPEEX_PREPROCESS_SET_NOISE_SUPPRESS Устанавливает максимальное подавление шума в дБ (отрицательное spx_int32_t)

SPEEX_PREPROCESS_GET_NOISE_SUPPRESS Возвращает максимальное подавление шума в дБ (отрицательное spx_int32_t)

SPEEX_PREPROCESS_SET_ECHO_SUPPRESS Устанавливает максимальное ослабление эха в дБ (отрицательное spx_int32_t)

SPEEX_PREPROCESS_GET_ECHO_SUPPRESS Возвращает максимальное ослабление эха в дБ (отрицательное spx_int32_t)

SPEEX_PREPROCESS_SET_ECHO_SUPPRESS_ACTIVE Устанавливает максимальное ослабление эха в дБ, когда когда ближний конец активен (отрицательное (negative spx_int32_t)

SPEEX_PREPROCESS_GET_ECHO_SUPPRESS_ACTIVE Get Возвращает максимальное ослабление эха в дБ, когда когда ближний конец активен (отрицательное spx_int32_t)

SPEEX_PREPROCESS_SET_ECHO_STATE Устанавливает связанный эхокомпенсатор для остаточного подавления эха (указатель или NULL для отключения подавления остаточного эха)

SPEEX_PREPROCESS_GET_ECHO_STATE Возвращает связанный эхокомпенсатор (указатель)

6.2. Подавление эха

Библиотека Speex на данный момент включает в себя алгоритм подавление эха подходящий для (AEC). В случае использования эхоподавления, в первую очередь необходимо:

```
#include <speex/speex_echo.h>
```

После чего режим эхоподавления должен быть установлен с помощью:

```
SpeexEchoState *echo_state = speex_echo_state_init(frame_size, filter_length);
```

где `frame_size` это количество данных (в выборках), которые необходимо вам обработать сразу, и `filter_lengt` это размер (в выборках) фильтра подавления эха который вы используете (также известен как *tail length*). Рекомендуется использовать размер кадра кратного 20 мс (или равного размеру кадра кодека), и убедитесь в том, что FFT легко выполняется при данном размере (степени двойки лучше чем остальные числа). Рекомендуемый размер фильтра приблизительно равен одной трети времени отражения звука в помещении. Например, в маленькой комнате, время отражения звука порядка 300 мс, таким образом размер фильтра равный 100 мс (800 выборок при частоте дискретизации в 8000 Гц) это правильный выбор.

После того, как режим подавления эха был остановлен, звук может отрабатываться с помощью:

```
speex_echo_cancellation(echo_state, input_frame, echo_frame, output_frame);
```

где `input_frame` это звук, который зафиксирован микрофоном, `echo_frame` это сигнал, который воспроизводится динамиками (и должен быть убран) и `output_frame` это сигнал с убранным эхо.

Одна важная вещь состоит в соотношении между `input_frame` и `echo_frame`. Важно то, что в любое время любое эхо, которое появляется на входе, было отправлено в фильтр эхоподавления как `echo_frame`. Другими словами, фильтр эхоподавления не может убрать сигнал, который еще не принят. С другой стороны, задержка между входным сигналом и сигналом эха должна быть достаточно малой, потому как иначе часть фильтра эхоподавления будет работать впустую. В идеальном случае ваша программа должна выглядеть так:

```
write_to_soundcard(echo_frame, frame_size);
read_from_soundcard(input_frame, frame_size);
speex_echo_cancellation(echo_state, input_frame, echo_frame, output_frame);
```

Если вы хотите в дальнейшем уменьшить эхо, представленное в сигнале, вы должны сделать это путем связывания фильтра эхоподавления с препроцессором (см. раздел 6.1). Это осуществляется с помощью вызова:

```
speex_preprocess_ctl(preprocess_state, SPEEX_PREPROCESS_SET_ECHO_STATE, echo_state);
```

при инициализации.

На момент версии 1.2-beta2, существует альтернативный, более простой API, который может быть использован вместо *speex_ echo_ cancellation*(). Когда захват звука и его воспроизведение обрабатывается асинхронно (т.е. в разных потоках или с использованием системных вызовов *poll()* или *select()*), тогда будет сложно отследить, который input_frame пришел с каким из *echo_frame*. Вместо этого, контекст или поток воспроизведения может быть просто вызван через:

```
speex_echo_playback(echo_state, echo_frame);
```

каждый раз когда воспроизводится звуковой кадр. После чего захват контекста/потока с помощью вызова:

```
speex_echo_capture(echo_state, input_frame, output_frame);
```

для каждого полученного кадра. Внутренне, *speex_ echo_ playback()* проще буферизирует воспроизводимый кадр, таким образом, этим можно воспользоваться, вызвав *speex_ echo_ capture()* для вызова *speex_ echo_ cancel()*. Побочный эффект использования альтернативного API состоит в том, что воспроизводимый звук задерживается на два кадра, что является нормальной задержкой, вызванной звуковой картой. Когда захват и воспроизведение синхронизированы, *speex_ echo_ cancellation()* предпочтительнее, так как предоставляет лучший контроль при точном входном/эхо тайминге.

Состояние эхоподавления может быть удалено с помощью:

```
speex_echo_state_destroy(echo_state);
```

Также можно сбросить состояние эхоподавления, таким образом, оно может быть использовано повторно без необходимости создания другого состояния, с помощью:

```
speex_echo_state_reset(echo_state);
```

6.2.1. Решение проблем

Существует несколько вещей, которые могут препятствовать правильной работе эхоподавления. Одна из них это ошибка (или нечто условно оптимальное) в коде, но существует множество других ошибок, на которых вы должны сосредоточиться в первую очередь.

- Использование разных звуковых карт для записи и воспроизведения **не** будет работать, независимо от того, что вы можете подумать. Единственное исключение состоит в том, что если две звуковые карты смогут зафиксировать их тактовые частоты на одном источнике тактовой частоты. Если нет, тогда тактовые частоты всегда будут иметь некоторый дрейф, который предотвратит адаптацию фильтра эхоподавления.
- Задержка между записью и воспроизведением сигналов должна быть минимальной. Любой воспроизводимый сигнал должен появиться на воспроизводящем устройстве (дальний конец) немного раньше, чем фильтр эхоподавления увидит его в записанном сигнале, но чрезмерная задержка подразумевает, что часть фильтра будет использовано впустую. В худших ситуациях задержка настолько большая, что больше чем длина фильтра, в данном случае, никакое эхо не будет подавляться.
- Когда дело доходит до длины фильтра, длиннее не значит лучше. На самом деле, при большей длине фильтра фильтр адаптируется дольше. Безусловно, слишком маленькая длина фильтра не достаточна для подавления эха, но наиболее распространенная проблема заключается в том что люди устанавливают слишком большую длину фильтра и после удивляются тому, что эхо не подавляется.
- Нелинейные искажения не могут быть (по определению) смоделированы линейным адаптивным фильтром, используемым в устройстве эхоподавления, и также не могут быть подавлены. Используйте хороший звуковой тракт и избежите насыщения/отсечения.

Также полезно прочесть Echo Cancellation Demystified Алексея Фрунзе[1], которая объясняет фундаментальные принципы эхоподавления. Детали алгоритма описанные в статье отличаются, но основные идеи эхоподавления через адаптивные фильтры одинаковы.

На момент версии 1.2beta2, новый инструмент `echo_diagnostic.m` включен в дистрибутив исходного кода. Первым шагом нужно определить DUMP_ECHO_CANCEL_DATA в процессе сборки. Это приводит к тому что фильтр эхоподавления автоматически сохраняет сигналы ближнего конца, дальнего конца и выходные сигналы в файлы (aec_rec.sw aec_play.sw и aec_out.sw). Это именно то что AEC получает и выдает. Для того чтобы им воспользоваться необходимо:

```
echo_diagnostic('aec_rec.sw', 'aec_play.sw', 'aec_diagnostic.sw', 1024);
```

Значение 1024 это длина фильтра которая может быть изменена. Имеют место быть некоторые (надеюсь) полезные выводимые сообщения и звук с подавленным эхом будет сохранен в aec_diagnostic.sw. Если даже эти выходные данные плохие (почти нет подавления) возможно имеется проблема в процессах воспроизведения или записи.

6.3. Буфер колебаний задержки

Буфер колебаний задержки может быть включен подключением:

```
#include <speex/speex_jitter.h>
```

и созданием состояния нового буфера колебаний задержки которое инициализируется путем:

```
JitterBuffer *state = jitter_buffer_init(step);
```

где аргумент step это время шага по умолчанию (в единицах временных меток) используется для регулирования задержек и реализации маскирования. Значение равное 1 всегда правильное, но более высокие значения могут быть иногда более удобными. Например, если вы способны делать сокрытие только на 20 миллисекундных кадрах, нет никакого смысла в буфере колебания задержки, делающего это с одной выборкой. Другой пример состоит в том, что для видео нет никакого смысла регулировать задержку меньшую, чем полный кадр. Предоставленное значение может быть изменено после.

API буфера колебаний задержки основано на типе JitterBufferPacket, который определен как:

```
typedef struct {
   char        *data;       /* Data bytes contained in the packet */
   spx_uint32_t len;        /* Length of the packet in bytes */
   spx_uint32_t timestamp;  /* Timestamp for the packet */
   spx_uint32_t span;       /* Time covered by the packet (timestamp units) */
} JitterBufferPacket;
```

[1] http://www.embeddedstar.com/articles/2003/7/article20030720-1.html

Например, для звука поле timestamp будет равняться тому, что содержится в поле timestamp RTP и в span будет количество выборок, закодированных в пакете. Для узкополосного режима, span будет равняться 160, если только один кадр включен в пакет.

Когда приходит пакет, необходимо вставить эти данные в буфер с помощью:

```
JitterBufferPacket packet;
/* Fill in each field in the packet struct */
jitter_buffer_put(state, &packet);
```

Когда декодер готов декодировать пакет, пакет может быть получен путем:

```
int start_offset;
err = jitter_buffer_get(state, &packet, desired_span, &start_offset);
```

Если jitter_buffer_put() и jitter_buffer_get() вызываются из различных потоков, тогда вам необходимо защитить состояние буфера колебаний задержки мьютексом.

Так как буфер колебаний задержки не создан для того, чтобы использовать таймер заданный в явном виде, необходимо сообщать о времени непосредственно. Это осуществляется путем вызова:

```
jitter_buffer_tick(state);
```

Это должно осуществляться периодически в воспроизводящем потоке. Это будет последним вызовом буфера колебаний задержки перед тем как уйти в сон (до тех пор пока больше данные не воспроизводятся). В некоторых случаях предпочтительнее использовать

```
jitter_buffer_remaining_span(state, remaining);
```

Второй аргумент используется для того чтобы указать на то, что мы до сих пор храним данные, записанные для воспроизводящего устройства. Например, если для звуковой карты требуется 256 выборок (определяется desired_span), но jitter_buffer_get() возвращает 320 выборок, у нас будет remaining=64.

6.4. Устройство передискретизации

Speex включает в себя модуль передискретизации. Для того чтобы им воспользоваться, необходимо подключить его заголовочный файл:

```
#include <speex/speex_resampler.h>
```

Для каждого потока, который должен быть передискретизирован, важно задать состояние устройства передискретизации с помощью:

```
SpeexResamplerState *resampler;
resampler = speex_resampler_init(nb_channels, input_rate, output_rate, quality, &err);
```

Где nb_channels это число каналов, которые будут использованы (или с чередованием или без чередования), input_rate это частота дискретизации во входном потоке, output_rate частота дискретизации в выходном потоке и quality это установка требуемого качества (от 0 до 10). Параметр качества важен для контроля компромисса между качеством сложностью и скрытостью. Использование более высоких установок качества обеспечивает меньший шум и сглаживание, более высокую сложность и большую скрытость. Использование параметра качества равного трем применимо для большинства пользователей ПК, качество 10 больше всего рекомендуется для профессиональной работы со звуком. Качество 0 обычно имеет приемлемый звук (обычно лучший чем полученный с использованием передискретизации линейной интерполяцией), но артефакты иногда могут быть заметны.

```
err = speex_resampler_process_int(resampler, channelID, in, &in_length, out, &out_length);
```

где channelID это ID канала который будет обрабатываться. Для монофонического потока используется 0. Указатель *in* указывает на первую выборку входного буфера выбранного канала и out указывает на первую выборку выходного буфера. Размер входного и выходного буфера определяется через in_length и out_length соответственно. После завершения эти значения заменяются количеством выборок прочитанных и записанных функцией передискретизации. Если не произойдёт ошибка или когда все входящие выборки будут прочитаны, или когда все исходящие выборки будут записаны (или в обеих случаях одновременно). Для выборок с плавающей точкой функцияspeex_resampler_process_float() ведет себя так же.

Также возможно обрабатывать несколько каналов одновременно. Для того, чтобы это сделать, используйте speex_resampler_process_interleaved_int() или speex_resampler_process_interleaved_float(). Аргументы такие же, за исключением отсутствия аргумента channelID. Стоит заметить что **параметры длины задаются для каждого канала**. То есть, если у вас 1024 выборки на каждый из четырех каналов, то необходимо передать 1024, а не 4096.

Передискретизатор позволяет менять качество и входные/выходные частоты дискретизации на лету без ошибок. Это может быть реализовано при помощи вызова функций speex_resampler_set_quality() и speex_resampler_set_ Единственный побочный эффект состоит в том, что новый фильтр будет вычислен повторно, потребляя множество циклов ЦПУ.

При передискретизации файла зачастую желательно иметь идеальную синхронизацию выходного файла с входным. Для того, чтобы это реализовать, вызовите speex_resampler_skip_zeros() **до начала** вычислений. Для приложений, работающих в реальном времени (т.е. VoIP), не рекомендуется этого делать, потому как первый кадр процесса будет короче из за компенсации задержки (пропущенные нули). Вместо этого, в приложениях реального времени вы можете узнать, насколько долгая задержка привнесена устройством передискретизации. Это может быть осуществлено во время работы программы при помощи функций speex_resampler_get_input_latency() и speex_resampler_get_output_latency() Первая функция возвращает задержку, измеренную в выборках при частоте дискретизации входного потока, в то время как вторая - в выборках, измеренных при частоте дискретизации выходного потока.

Для того чтобы уничтожить передискретизатор, вызовите speex_resampler_destroy().

6.5. Циклический буфер

В некоторых случаях необходимо согласование с компонентами которые используют разные размеры блоков. Например возможно, что звуковая карта не поддерживает чтение/запись блоками по 20 мс или иногда, сложные соотношения передискретизации означают, что блоки не всегда имеют одинаковое время. В таких случаях необходимо сохранить данные используя циклический буфер.

7. Форматы и стандарты

Speex может кодировать речь в обеих узкополосном и широкополосном режимах и обеспечивая разные битрейты. Хотя не все функции необходимы для поддержки в определенной реализации или определенном устройстве. Для того, чтобы называться совместимой со Speex, реализация должна поддерживать основной набор функций.

Как минимум, все узкополосные режимы работы ДОЛЖНЫ быть поддерживаемы декодировщиком. Это включает в себя декодирование широкополосного потока узкополосным декодировщиком Широкополосный битовый поток содержит узкополосный битовый поток, который может быть декодирован отдельно[1]. Таким образом, декодировщик ОБЯЗАН кодировать узкополосный поток, и МОЖЕТ либо декодировать все широкополосные режимы, либо декодировать встроенную узкополосную часть всех режимов (которое включает в себя игнорирование битов высоких частот).

Для кодировщиков, хотя бы один узкополосный или широкополосный режим ДОЛЖЕН поддерживаться. Основная причина, по которой все режимы кодирования могут не поддерживаться состоит в том, что некоторые платформы не смогут обработать сложный кодирующий алгоритм некоторых режимов.

7.1. Формат данных RTP

Проект полезной нагрузки в RTP включен в приложении C и последняя версия доступна в `http://www.speex.org/drafts/latest`. Этот проект был отправлен (2003/02/26) в Internet Engineering Task Force (IETF) и будет обсуждаться на встрече 18 марта в Сан-Франциско.

7.2. Тип MIME

Теперь вы можете использовать тип MIME audio/x-speex для Speex-in-Ogg. Мы подадим заявку для типа audio/speex в ближайшем будущем.

7.3. Формат файла ogg

Поток данных Speex может храниться в файлах Ogg. В данном случае, первый пакет Ogg файла содержит заголовок Speex, описанный в таблице 7.1. Все целочисленные поля в заголовках хранятся с прямым порядком битов. Поле `speex_string` должно содержать "`Speex   `" (с тремя пробелами), которые определяют битовый поток. Следующее поле, `speex_version` содержит версию Speex которая кодировала файл. На данный момент, обратитесь к speex_header.[ch] для дополнительной информации. *Флаг начала потока* (`b_o_s`) выставляется в 1 для заголовка. Заголовок пакета имеет `packetno=0` и `granulepos=0`.

Второй пакет содержит заголовок комментария Speex. Используемый формат, это формат комментария Vorbis, описанный здесь: http://www.xiph.org/ogg/vorbis/doc/v-comment.html . Данный пакет имеет `packetno=1` и `granulepos=0`.

Третий и последующие пакеты каждый содержат один или более (количество указанное в заголовке) кадров Speex. Они идентифицируются при помощи packetno начинающегося с 2 и `granulepos` - количеством последних выборок, закодированных в пакете. У последнего из этих пакетов флаг *end of stream* (`e_o_s`) установлен в единицу.

[1] The wideband bit-stream contains an embedded narrowband bit-stream which can be decoded alone

Поле	Тип	Размер
speex_string	char[]	8
speex_version	char[]	20
speex_version_id	int	4
header_size	int	4
rate	int	4
mode	int	4
mode_bitstream_version	int	4
nb_channels	int	4
bitrate	int	4
frame_size	int	4
vbr	int	4
frames_per_packet	int	4
extra_headers	int	4
reserved1	int	4
reserved2	int	4

Table 7.1.: Заголовок пакета Ogg/Speex

8. Введение в CELP

Не вмешивайтесь в дела полюсов, ибо непостижимы они и стремятся покинуть единичную окружность.

Speex основан на CELP, что означает линейное предсказание с мультикодовым управлением. В этом разделе предполагается введение в основные принципы CELP, так что, если вы знакомы с CELP, вы можете совершенно спокойно пропустить этот раздел 10. Алгоритм CELP базируется на трех основных идеях:

1. Использование модели линейного предсказания в качестве модели голосового тракта
2. Использование (адаптивной или фиксированной) кодовой таблицы выступающей в качестве источника модели линейного предсказания (ЛП)
3. Поиск осуществляется в замкнутом цикле в “перцепционно взвешенном домене”

Данный раздел описывает основные идеи, используемые в CELP. Раздел по прежнему в стадии разработки.

8.1. Модель предсказания речи "источник-фильтр"

Данная модель подразумевает, что голосовые связки являются источниками спектрально плоского звука, и что голосовой тракт играет роль фильтра по отношению к спектральной форме различных звуков речи. Пока еще аппроксимация является широко используемой моделью в кодировании речи по причине ее простоты. Ее использование является также причиной, по которой большинство речевых кодеков (в том числе и Speex) плохо работают при кодировании музыки. Различные фонемы могут быть выделены по их возбудителю (источнику) и спектральной форме (фильтру). У гласных фонем источник сигнала является периодическим и может быть аппроксимирован импульсной последовательностью во временной плоскости или гармониками, расположенными через определенные промежутки, в частотной области. С другой стороны, глухие согласные (такие как "с", "ш", "ф") имеют источник сигнала, который похож на белый гауссовский шум. Также звонкие согласные (такие как "з", "в") имеют сигнал возбуждения, состоящий из гармонической и шумовой части.

Модель источник-фильтр обычно связана с использованием линейного предсказания. Модель CELP основана на модели источник-фильтр, как это можно увидеть на CELP декодировщике, показанном на диаграмме ниже 8.1.

8.2. Коэффициенты линейного предсказания (LPC)

Линейное предсказание лежит в основе множества техник кодирования речи, включая CELP. Идея заключается в том, что сигнал $x[n]$ предсказывается с использованием линейной комбинации его выборок:

$$y[n] = \sum_{i=1}^{N} a_i x[n-i]$$

где $y[n]$ линейное предсказание $x[n]$. Таким образом, ошибка предсказания представлена как:

$$e[n] = x[n] - y[n] = x[n] - \sum_{i=1}^{N} a_i x[n-i]$$

Цель линейного предсказания в том, чтобы найти наилучшие коэффициенты предсказания a_i, при которых значение функции среднеквадратичной ошибки является наименьшей:

$$E = \sum_{n=0}^{L-1} [e[n]]^2 = \sum_{n=0}^{L-1} \left[x[n] - \sum_{i=1}^{N} a_i x[n-i] \right]^2$$

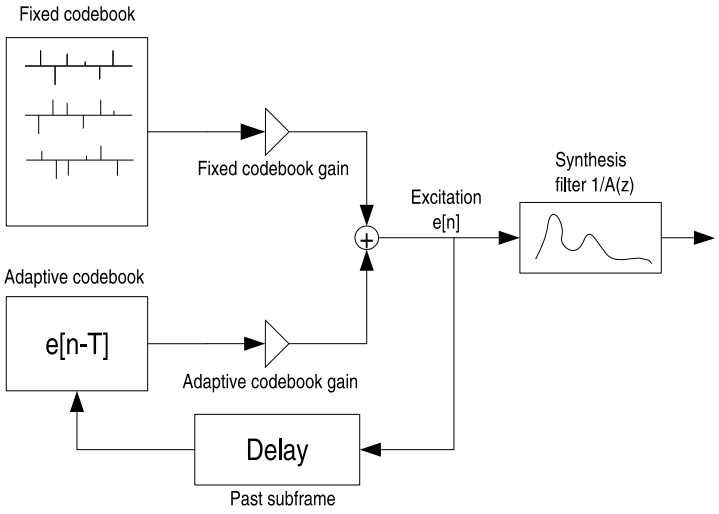


Figure 8.1.: Модель синтеза речи CELP (декодер)

Это может быть осуществлено путем зануления производных $\frac{\partial E}{\partial a_i}$:

$$\frac{\partial E}{\partial a_i} = \frac{\partial}{\partial a_i} \sum_{n=0}^{L-1} \left[x[n] - \sum_{i=1}^{N} a_i x[n-i] \right]^2 = 0$$

Для фильтра порядка N коэффициенты a_i могут быть найдены с помощью решения системы линейных уравнений $\mathbf{Ra} = \mathbf{r}$ размерностью $N \times N$, где

$$\mathbf{R} = \begin{bmatrix} R(0) & R(1) & \cdots & R(N-1) \\ R(1) & R(0) & \cdots & R(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ R(N-1) & R(N-2) & \cdots & R(0) \end{bmatrix}$$

$$\mathbf{r} = \begin{bmatrix} R(1) \\ R(2) \\ \vdots \\ R(N) \end{bmatrix}$$

с $R(m)$, автокорреляцией сигнала $x[n]$, вычисляемой по следующей формуле:

$$R(m) = \sum_{i=0}^{N-1} x[i]x[i-m]$$

Так как $\mathbf{R}$ самосопряженная матрица Тёплица, для решения данной системы уравнений может быть использован алгоритм Левинсона-Дарбина , решающий эту задачу за $\mathcal{O}\left(N^2\right)$ итераций вместо $\mathcal{O}\left(N^3\right)$. Также может быть доказано что все корни $A(z)$ находятся в пределах единичного круга, это означает, что $1/A(z)$ всегда стабилен. Но это в теории, на практике это не всегда является справедливым по причине конечной точности, существуют две часто используемых техники, подтверждающие стабильность фильтра. Во первых, можно умножить $R(0)$ на число немного большее единицы (такое как 1.0001), что, в свою очередь, эквивалентно добавлению шума к сигналу. Также мы можем применить оконную фильтрацию к автокорреляции, что эквивалентно фильтрации в частотной области, уменьшающей острые резонансы.

8.3. Предсказание тембра

В течении звонких сегментов голосовой сигнал периодический, таким образом, можно использовать это свойство путем аппроксимации сигнала возбуждения $e[n]$, представляя его как усиленный в β раз предыдущий

сигнал:

During voiced segments, the speech signal is periodic, so it is possible to take advantage of that property by approximating the excitation signal $e[n]$ by a gain times the past of the excitation:

$$e[n] \simeq p[n] = \beta e[n-T] \ ,$$

где T период основного тона, β усиление основного тона. Мы называем это долгосрочным предсказанием если источник предсказывается как $e[n-T]$ с $T \gg N$.

8.4. Инновационная кодовая таблица

Последний сигнал возбуждения $e[n]$ будет являться суммой предсказанного тона и инновационного сигнала $c[n]$, взятого из фиксированной кодовой таблицы, отсюда и название линейное предсказание с мультикодовым управлением. Последнее выражение имеет вид:

$$e[n] = p[n] + c[n] = \beta e[n-T] + c[n] \ .$$

Квантование $c[n]$, где расположено большинство бит в CELP. Представляет информацию, которая не может быть получена ни через линейное предсказание ни через предсказание тона. В Z плоскости мы можем представить итоговый сигнал $X(z)$ как

$$X(z) = \frac{C(z)}{A(z)\,(1-\beta z^{-T})}$$

8.5. Взвешивание шума

Большинство (если не вообще все) современных звуковых кодеков пытается сформировать шум так, чтобы он отображался, в основном, в тех частотных областях, в которых его не сможет распознать слух человека. Например, слух меньше замечает шус в тех областях спектра, которые громче и *наоборот*. Для того, чтобы максимизировать качество речи, CELP кодеки минимизируют среднеквадратичную ошибку в перцепционно взвешенном домене. Это означает, что перцепционно взвешенный шумовой фильтр $W(z)$ применяется к сигналу ошибки в кодировщике. В большинстве CELP кодеков $W(z)$ это взвешивающий фильтр, полученный из коэффициентов линейного предсказания, как правило с помощью расширения полосы частот. Предположим, что огибающая спектра представлена синтезирующим фильтром $1/A(z)$, тогда в CELP кодеках полученный взвешивающий фильтр шумов обычно представлен как:

$$W(z) = \frac{A(z/\gamma_1)}{A(z/\gamma_2)} \ , \tag{8.1}$$

где $\gamma_1 = 0.9$ и $\gamma_2 = 0.6$ в базовой реализации Speex. Если фильтр $A(z)$ имеет комплексные полюса p_i на z-плоскости, фильтр $A(z/\gamma)$ будет иметь полюса $p'_i = \gamma p_i$, таким образом добиваясь более плоского варианта $A(z)$.

Взвешивающий фильтр применяется к сигналу ошибок для оптимизации поиска по кодовой таблице при анализе через синтез (Analysis-by-Synthesis или AbS). Это приводит к определенному результату в спектре шума, который стремится к $1/W(z)$. Пока простота модели является важной причиной успеха CELP, остается, что $W(z)$ это очень грубая аппроксимация для функции перцепционно оптимального взвешивания шума. Рисунок 8.2 показывает сглаживание шума, полученное с помощью выражения 8.1. Во всем данном руководстве под $W(z)$ мы подразумеваем взвешивающий фильтр шума и под $1/W(z)$ мы подразумеваем формирующий шумовой фильтр (или кривая).

8.6. Анализ через синтез

Один из основных принципов CELP назван анализом через синтез (AbS), это означает, что кодирование (анализ) осуществляется с помощью перцепционной оптимизации декодированного (синтез) сигнала в замкнутом цикле. В теории лучший CELP поток должен быть произведен путем проверки всех возможных комбинаций битов и выбора одной, которая воспроизведет декодированный сигнал, звучащий наилучшим образом. Очевидно, что это не реализуемо на практике по двум причинам: требуемая сложность намного выше возможностей

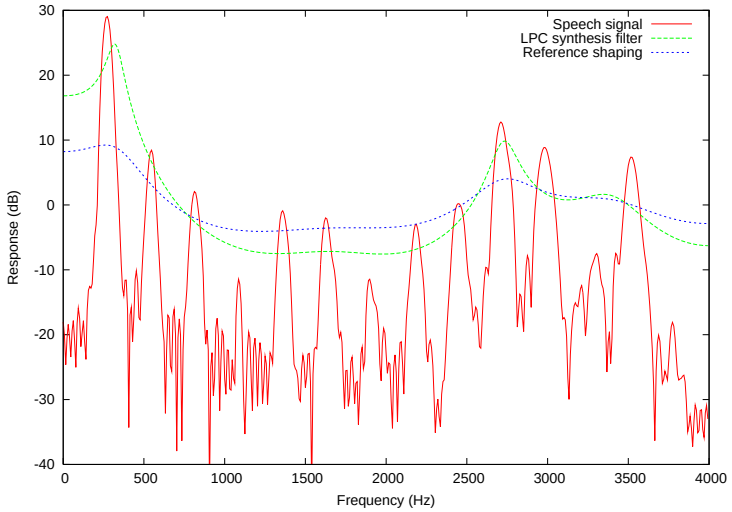


Figure 8.2.: Стандартное подавление шума в CELP. Произвольное смещение оси y.

доступного на данный момент аппаратного обеспечения и критерий выбора наилучшего звучания определяется слушателем.

С целью достижения кодирования в реальном времени с использованием ограниченных вычислительных ресурсов, оптимизация разбивается на меньшие, более управляемые, последовательные поиски с использованием взвешивающей функции, описанной ранее.

9. Спецификация декодировщика Speex

Это незаконченный раздел, перевел дословно, хоть посмеётесь

9.1. Узкополосный декодер

<Вставьте сюда блок-схему декодера>

9.1.1. Узкополосные режимы

Существует 7 разных узкополосных битрейтов определенных для Speex, в диапазоне от 250 бит/с до 24.6 кбит/с, хотя режимы меньше 5.9 кбит/с не могут быть использованы для речи. Битовая разметка для каждого режима представлена в таблице 9.1. Каждый фрейм с режимом ID кодируется 4 битами предоставляющзими диапазон от 0 до 15, хотя используются только первые 7 значений (остальные зарезервированы). Параметры перечислены в таблице в том порядке, в котором они упакованы в битовом потоке. Все параметры кадров упакованы до параметров субкадров. Параметры для определенного субкадра упаковываются до следующего субкадра. "OL" в описании параметра означает что параметр основан на оценке в открытом цикле (open loop) всего кадра.

Параметр	Частота обновления	0	1	2	3	4	5	6	7	8
Широкополосный бит	кадр	1	1	1	1	1	1	1	1	1
ID режима	кадр	4	4	4	4	4	4	4	4	4
ЛСП	кадр	0	18	18	18	18	30	30	30	18
OL pitch	кадр	0	7	7	0	0	0	0	0	7
OL pitch gain	кадр	0	4	0	0	0	0	0	0	4
OL Exc gain	кадр	0	5	5	5	5	5	5	5	5
Fine pitch	субкадр	0	0	0	7	7	7	7	7	0
Pitch gain	субкадр	0	0	5	5	5	7	7	7	0
Инновационный gain	субкадр	0	1	0	1	1	3	3	3	0
инновационый VQ	субкадр	0	0	16	20	35	48	64	96	10
Итог	кадр	5	43	119	160	220	300	364	492	79

Table 9.1.: Битовая разметка для узкополосных режимов

9.1.2. Декодирование ЛСП

В зависимости от режима, параметры кодируются 18 или 30 битами.
Интерполяция
Safe margin

9.1.3. Адаптивная кодовая таблица

Для скоростей в 8 кбит/с и выше период шага кодируется для каждого субфрейма. Реальный период равен $T = p_i + 17$ где p_i число кодируемое 7 битами и 17 соотвуют минимальному шагу. Максимальный период 144. На 5.95 кбит/с (режим 2), период шага кодируется похожим образом, но только один раз за кадр. Каждый субкадр имеет 2 битовое смещение, которое добавляется к значению шага кадра. В этом случае шаг для каждого субкадра равен $T - 1 + offset$. Для битрейтов ниже 5.95 кбит/с используется только кадровый шаг и шаг постоянен для всех субкадров.

Speex использует предсказание с тремя коэффициентами для битрейтов 5.95 кбит/с и выше. Три коэффициента усиления извлекаются из 5 битной и 7 битной кодовой таблицы в зависимости от режима.

9.1.4. Инновационная кодовая таблица

Разбивка кодовой таблицы, размер и записи зависят от битрейта
5 битный коэффициент это кодер на основе кадра
В зависимости от режима, большее разрешение на кадый субкадр
инновационныые субвекторы склеиваются, коэффициент применяется

9.1.5. Перцепционный улучшайзинг

Опционально, в зависимости от приложенеия.

9.1.6. Спецификация битового потока

Этот раздел определяет битовый поток который передается по проводам. Обин пакет speex состоит из 1 заголовка кадра и 4 субкадров:

Заголовок кадра	Субкадр 1	Субкадр 2	Субкадр 3	Субкадр 4

Заголовок кадра имеет переменную длину, в зависимости от режима декодирования и подрежима. Заголовок кадра узкополосного режима определен следующим образом:

wb bit	modeid	ЛСП	OL-pitch	OL-pitchgain	OL ExcGain

wb-bit: Узкополосный бит (1 бит) 0=узкополосный, 1=широкополосный
modeid: Идентификатор режима (4 бита)
ЛСП: Линейные спектральные пары (0, 18 or 30 бит)
OL-pitch: Шаг открытого цикла (0 or 7 бит)
OL-pitchgain: Коэффициент усиления шага открытого цикла(0 or 4 бит)
OL-ExcGain: Коэффициент усиление сигнала возбуждения открытого цикла (0 or 5 бит)
...
Каждый подкадр определен как:

FinePitch	PitchGain	InnovationGain	Innovation VQ

FinePitch: (0 or 7 бит)
PitchGain: (0, 5, or 7 бит)
Innovation Gain: (0, 1, 3 бита)
Innovation VQ: (0-96 бит)
...

9.1.7. Декодер сэмплов

Этот раздел содержит некоторый исходный код с семлами, показывающий как может быть написан базовый декодер Speex. Базовый декодер Speex это только третий субмодуль узкополосного режима без сложных функций таких как улучшайзинг, vbr, т.д.
...

Listing 9.1: Sample Decoder

```
#include <math.h>
#include "nb_celp.h"
#include "lsp.h"
#include "ltp.h"
#include "quant_lsp.h"
#include "cb_search.h"
#include "filters.h"
#include "os_support.h"

#ifndef NULL
#define NULL 0
#endif
```

```
#define LSP_MARGIN .002f
#define SIG_SCALING  1.f
#define NB_DEC_BUFFER (NB_FRAME_SIZE+2*NB_PITCH_END+NB_SUBFRAME_SIZE+12)
#define NB_ORDER 10
#define NB_FRAME_SIZE 160
#define NB_SUBFRAME_SIZE 40
#define NB_NB_SUBFRAMES 4
#define NB_PITCH_START 17
#define NB_PITCH_END 144


struct speex_decode_state {
        float excBuf[NB_DEC_BUFFER]; /**< Excitation buffer */
        float *exc;                  /**< Start of excitation frame */
        float old_qlsp[10];          /**< Quantized LSPs for previous frame */
        float interp_qlpc[10];       /**< Interpolated quantized LPCs */
        float mem_sp[10];            /**< Filter memory for synthesis signal */
        int first;                   /**< Is this the first frame? */
};


static const float exc_gain_quant_scal1[2] = {0.70469f, 1.05127f};


struct speex_decode_state *nb_decoder_init(void)
{
        struct speex_decode_state *st;

        st = malloc(sizeof(*st));
        if (!st)
                return NULL;

        memset(st, 0, sizeof(*st));
        st->first = 1;

        return st;
}


void nb_decoder_destroy(struct speex_decode_state *state)
{
        if (state)
                free(state);
}


/* basic decoder using mode3 only */
int nb_decode(struct speex_decode_state *st, SpeexBits *bits, float *out)
{
        int i, sub, wideband, mode, qe;
        float ol_gain;
        float innov[NB_SUBFRAME_SIZE];
        float exc32[NB_SUBFRAME_SIZE];
        float qlsp[NB_ORDER], interp_qlsp[NB_ORDER];
        float ak[NB_ORDER];
```

```
if (!bits)
        return -1;

st->exc = st->excBuf + 2*NB_PITCH_END + NB_SUBFRAME_SIZE + 6;

/* Decode Sub-modes */
do {
        if (speex_bits_remaining(bits) < 5)
                return -1;

        wideband = speex_bits_unpack_unsigned(bits, 1);
        if (wideband) {
                printf("wideband not supported\n");
                return -2;
        }

        mode = speex_bits_unpack_unsigned(bits, 4);
        if (mode == 15)
                return -1;

} while (mode > 8);

if (mode != 3) {
        printf("only mode 3 supported\n");
        return -2;
}

/* Shift all buffers by one frame */
SPEEX_MOVE(st->excBuf, st->excBuf+NB_FRAME_SIZE,
           2*NB_PITCH_END + NB_SUBFRAME_SIZE + 12);

/* Unquantize LSPs */
lsp_unquant_lbr(qlsp, NB_ORDER, bits);

/* Handle first frame */
if (st->first) {
        st->first = 0;

        for (i=0; i<NB_ORDER; i++)
                st->old_qlsp[i] = qlsp[i];
}

/* Get global excitation gain */
qe = speex_bits_unpack_unsigned(bits, 5);
ol_gain = SIG_SCALING*exp(qe/3.5);

/* Loop on subframes */
for (sub=0; sub<4; sub++) {
        int offset, q_energy;
        float *exc, *sp;
        float ener;

        offset = NB_SUBFRAME_SIZE*sub;
        exc = st->exc + offset;
        sp = out + offset;
```

```
            SPEEX_MEMSET(exc, 0, NB_SUBFRAME_SIZE);

            /* Adaptive codebook contribution */
            pitch_unquant_3tap(exc, exc32, NB_PITCH_START,
                               NB_SUBFRAME_SIZE, bits, 0);

            sanitize_values32(exc32, -32000, 32000, NB_SUBFRAME_SIZE);

            /* Unquantize the innovation */
            SPEEX_MEMSET(innov, 0, NB_SUBFRAME_SIZE);

            /* Decode sub-frame gain correction */
            q_energy = speex_bits_unpack_unsigned(bits, 1);
            ener = exc_gain_quant_scal1[q_energy] * ol_gain;

            /* Fixed codebook contribution */
            split_cb_shape_sign_unquant(innov, bits);

            /* De-normalize innovation and update excitation */
            signal_mul(innov, innov, ener, NB_SUBFRAME_SIZE);

            for (i=0; i<NB_SUBFRAME_SIZE; i++) {
                    exc[i] = exc32[i] + innov[i];
            }
    }

    SPEEX_COPY(out, &st->exc[-NB_SUBFRAME_SIZE], NB_FRAME_SIZE);

    /* Loop on subframes */
    for (sub=0; sub<4; sub++) {
            const int offset = NB_SUBFRAME_SIZE*sub;
            float *sp, *exc;

            sp = out + offset;
            exc = st->exc + offset;

            /* LSP interpolation (quantized and unquantized) */
            lsp_interpolate(st->old_qlsp, qlsp, interp_qlsp, NB_ORDER,
                            sub, NB_NB_SUBFRAMES, LSP_MARGIN);

            /* Compute interpolated LPCs (unquantized) */
            lsp_to_lpc(interp_qlsp, ak, NB_ORDER);

            iir_mem16(sp, st->interp_qlpc, sp, NB_SUBFRAME_SIZE,
                      NB_ORDER, st->mem_sp);

            /* Save for interpolation in next frame */
            for (i=0; i<NB_ORDER; i++)
                    st->interp_qlpc[i] = ak[i];
    }

    /* Store the LSPs for interpolation in the next frame */
    for (i=0; i<NB_ORDER; i++)
            st->old_qlsp[i] = qlsp[i];
```

```
        return 0;
}
```

9.1.8. Таблицы кодировки

Декодировщик Speex включает в себя набор таблиц кодировки и кодовых таблиц, которые используются для конвертирования между значениями в разных величинах. Это включает в себя:

- Корэффициенты возбуждения 10x16 (3200 бит/с)
- Корэффициенты возбуждения 10x32 (4000 бит/с)
- Корэффициенты возбуждения 20x32 (2000 бит/с)
- Корэффициенты возбуждения 5x256 (12800 бит/с)
- Корэффициенты возбуждения 5x64 (9600 бит/с)
- Корэффициенты возбуждения 8x128 (7000 бит/с)
- Кодовая таблица для трехступенчатого коэффициента усиления шага предсказания (Нормальный и низкий битрейт)
- Кодовая таблица для ЛСП в узкополосном резиме CELP
- ...

Эти таблицы кодировки приведены здесь для прикола.

exc_5_64_table.c
exc_5_256_table.c
exc_8_128_table.c
exc_10_16_table.c
exc_10_32_table.c
exc_20_32_table.c
gain_table.c
gain_table_lbr.c
lsp_tables_nb.c

9.2. Широкополосный декодер

QMF фильтр. Узкополосный сигнал декодируется с использованием узкополосного декодера

Для диапазонов выше, декодер похож на узкополосный с небольшим отличием, состоящим в том что здесь нету адаптивной кодовой таблицы.

Коэффициент усиления не кадый субкадр

10. Узкополосный режим Speex

Данный раздел рассматривает то, как работает Speex в узкополосном (частота дискретизации 8 kHz) режиме. Размер кадра для данного режима равен 20 ms, что соответствует 160 выборкам. Каждый кадр разделяется на 4 подкадра по 40 выборок каждый.

Также большинство решений дизайна было основано на следующих оригинальных целях и предположениях:

- Минимизация количества информации, извлекаемой из предыдущих кадров (для стойкости к потере пакетов)
- Динамически выбираемые кодовые таблицы (LSP, шаговые и инновационные)
- Кодовые таблицы с фиксированным субвектором (инновационные)

10.1. Анализ кадра целиком

В узкополосном режиме, кадры Speex длиной в 20 мс (160 выборок) и разделены на 4 субкадра по 5 мс каждый (40 выборок). Для большинства узкополосных битрейтов (8 кбит/с и выше) параметры, кодируемые только на уровне кадра, это линейные спектральные пары (ЛСП) и глобальное усиление сигнала возбуждения g_{frame}, как показано на рисунке 10.1. Все остальные параметры кодируются на уровне субкадров.

Анализ линейного предсказания осуществляется один раз за кадр с использованием асимметричного окна Хемминга, центрированного по четверти субкадра. Потому что коэффициенты линейного предсказания (КЛП) квантуются не сразу, сначала они конвертируются в линейные спектральные пары (ЛСП). Считается что коэффициенты ЛСП связаны с четвертью субкадров, и коэффициенты, связанные с первыми тремя субкадрами, линейно интерполируются с использованием текущего и предыдущего коэффициента ЛСП. Полученные коэффициенты конвертируются обратно в коэффициенты КЛП фильтра $\hat{A}(z)$. Не квантованный интерполированный фильтр представлен как $A(z)$ и может быть использован для взвешивающего фильтра $W(z)$, потому как нет необходимости в доступности данного фильтра для декодировщика.

Для того, чтобы сделать Speex более стойким к потере пакетов, к коэффициентам ЛСП не применяется предсказание до квантования. Коэффициенты ЛСП кодируются с помощью векторного квантования (VQ) с 30 битами для более высокого качества и 18 битами для более низкого качества.

10.2. Анализ через синтез субкадров

Цикл кодировщика анализа через синтез показан на рисунке 10.2. Существуют три основных аспекта по которым Speex значительно отличается от большинства остальных CELP кодировщиков. Во-первых, пока большинство новых CELP кодировщиков используют дробную оценку тембра с одним коэффициентом усиления, Speex использует целое число для кодирования периода тембра, но используя предсказание с тремя коэффициентами усиления. Роль адаптивной кодовой таблицы $e_a[n]$ может быть представлено как:

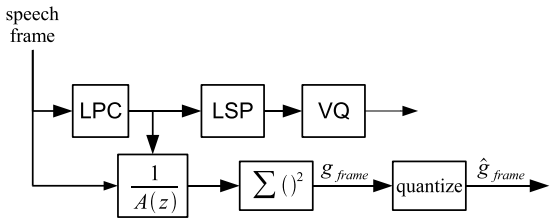


Figure 10.1.: Frame open-loop analysis

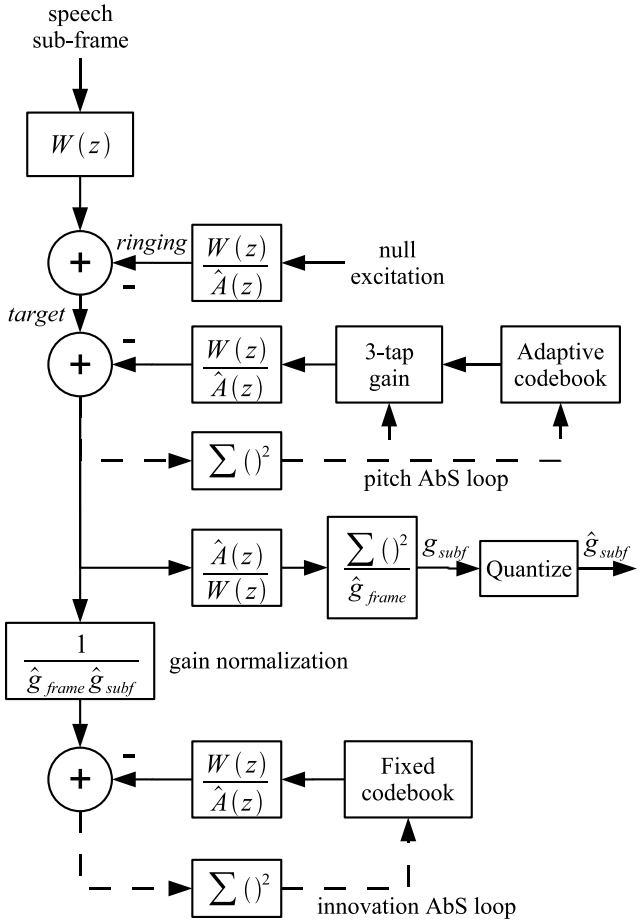


Figure 10.2.: Оптимизация субкадра анализом через синтез в замкнутом цикле.

$$e_a[n] = g_0 e[n-T-1] + g_1 e[n-T] + g_2 e[n-T+1] \quad (10.1)$$

где g_0, g_1 и g_2 это совместно квантованные коэффициенты усиления тембра и $e[n]$ это сигнал возбуждения в памяти кодека. Стоит заметить, что когда длина участка с одинаковым тембром меньше длины субкадра, мы повторяем сигнал возбуждения на период T. Например, когда $n-T+1 \geq 0$, мы используем $n-2T+1$ вместо этого. В большинстве режимов период тембра кодируется семью битами в диапазоне $[17, 144]$ и коэффициенты β_i квантованы векторно с 7 битами на более высоких битрейтах (15 кбит/с в узкополосном режиме и более) и пятью на более низких (11 кбит/с в узкополосном режиме и менее).

Большинство существующих на данный момент CELP кодеков используют предсказание со скользящим средним (MA) для кодирования коэффициента усиления из фиксированной кодовой таблицы. Это предоставляет немного лучшее кодирование ценой введения зависимости от ранее закодированных кадров. Второе отличие состоит в том, что Speex кодирует коэффициент усиления как продукт глобального усиления сигнала возбуждения g_{frame} с поправками усиления субкадра g_{subf}. Это увеличивает стойкость к потере пакетов путем уничтожения межкадровой зависимости. Коэффициент усиления субкадра кодируется до поиска в фиксированной таблице (оптимизация без обратной связи) и использует между 0 и 3 битами на субкадр, в зависимости от битрейта.

Третье различие состоит в том, что Speex использует субвекторное квантование для инновационного (с фиксированной таблицей) сигнала вместо алгебраической кодовой таблицы. Каждый субкадр разделен на субвекторы длины, варьирующейся между 5 и 20 выборками. Каждый субвектор выбирается из кодовой таблицы, зависимой от битрейта, и все субвекторы склеиваются для того, чтобы сформировать субкадр. В качестве примера, режим с 3.95 кбит/с использует размер субвектора в 20 выборок с 32 записями в кодовой таблице (5 бит). Это означает, что инновационный сигнал кодируется с 10 битами на субкадр, или 2000 бит/с. С другой стороны, в режиме 18.2 кбит/с используется размер субвектора из 5 выборок с 256 записями в кодовой таблице (8 bits), то есть инновационный сигнал использует 64 бита на субкадр, или 12800 бит/с.

10.3. Битрейты

До сих пор нет MOS (абсолютно субъективного понятия), но субъективная оценка была проведена для Speex. Для того, чтобы дать общее представление о качестве, достижимом с помощью данного кодека, таблица 10.1 представляет мое субъективное мнение об этом. Следует заметить, что разные люди воспринимают качество по разному и человек, создавший этот кодек, имел некоторую предвзятость при субъективной оценке. Последнее, что следует упомянуть, это то, что качество большинства кодеков (включая Speex) варьируется в зависимости от входного сигнала. Обратите внимание, что сложность только приблизительная (с 0.5 мфлопс и использованием наименьшей установки сложности). Для декодирования требуется приблизительно 0.5 мфлопс в большинстве режимов (1 мфлопс с перцепционным усилением).

Режим	Качество	Битрейт (бит/с)	мфлоп	Quality/description
0	-	250	0	Нет передачи (DTX)
1	0	2,150	6	Вокодер (в основном для комфортного шума)
2	2	5,950	9	Очень заметные артефакты/шум, хорошая разборчивость
3	3-4	8,000	10	Артефакты/шум иногда заметны
4	5-6	11,000	14	Артефакты иногда заметны только в наушниках
5	7-8	15,000	11	Нужны хорошие наушники для того, чтобы заметить разни
6	9	18,200	17.5	Трудно заметить разницу даже в хороших наушниках
7	10	24,600	14.5	Полностью прозрачен для голоса, хорошее качество музык
8	1	3,950	10.5	Очень заметные артефакты/шум, хорошая разборчивость
9	-	-	-	зарезервировано
10	-	-	-	зарезервировано
11	-	-	-	зарезервировано
12	-	-	-	зарезервировано
13	-	-	-	Определяется реализацией, интерпретируется вызовом функции или п
14	-	-	-	Внутриполосная связь Speex
15	-	-	-	Код разделителя

Table 10.1.: Качество в сравнении с битрейтом

10.4. Перцепционное усиление

Данный раздел действителен только для версии 1.1.12 и более ранних. Он не применим к версии 1.2-beta1 (и поздних), для которых новое перцепционное усиление до сих пор не задокументировано.

Данная часть кодека применяется только к декодировщику и даже может быть изменена без нарушения функциональной совместимости. По этой причине, реализация предоставленная и описанная здесь должна рассматриваться только как пример реализации. Усиление разделено на две части. Первая, это синтезирующий фильтр $S(z) = 1/A(z)$ заменен улучшенным фильтром:

$$S'(z) = \frac{A\left(z/a_2\right) A\left(z/a_3\right)}{A\left(z\right) A\left(z/a_1\right)}$$

где a_1 и a_2 зависят от используемого режима и a_{3}=\frac{1}$a_3 = \frac{1}{r}\left(1 - \frac{1-ra_1}{1-ra_2}\right)$ с $r = .9$. Вторая часть усиления состоит из использования гребенчатого фильтра для того, чтобы улучшить шаг в области сигнала возбуждения.

11. Широкополосный режим Speex (поддиапазон CELP)

Для широкополосного режима, Speex использует *квадратурный зеркальный фильтр* (QMF) для того чтобы расколоть диапазон на два. Таким образом сигнал частотой 16 кГц разделяется на два сигнала частотой 8 кГц, один представляет нижний диапазон (0-4 кГц), другой верхний диапазон (4-8 кГц). Нижний диапазон кодируется в узкополосном режиме, описанном в разделе 10, таким образом, полученный "встроенный узкополосный поток" также может быть декодирован узкополосным декодировщиком. Т. к. кодирование нижнего диапазона уже описано, в данном разделе будет рассматриваться только кодирование верхнего диапазона.

11.1. Линейное предсказание

Линейное предсказание для верхнего диапазона очень похоже на то, что использовалось для узкополосного режима. Единственная разница в, том что мы используем только 12 бит для кодирования LPS верхнего поддиапазона, используя многоступенчатый векторный квантователь (MSVQ). Первый уровень квантуется 10 коэффициентами с 6 битами и ошибкой, которая после квантуется с использованием так же 6 бит.

11.2. Предсказание тембра

Эта часть описания простая: нет никакого предсказания тембра для верхнего поддиапазона. На то существуют две причины. Во-первых, гармоническая структура обычно слабо выражена в этом диапазоне (выше 4 кГц). Во-вторых, это было бы очень трудно реализовать так как QMF загибает диапазон 4-8 кГц в 4-0 кГц (переворачивая частотную ось), это означает, что гармоники больше не расположены на частотах кратных фундаментальным (тембру).

11.3. Квантование сигнала возбуждения

Сигнал возбуждения верхнего диапазона кодируется так же, как и сигнал в узкополосном режиме.

11.4. Битовая разметка

Для широкополосного режима внутренний узкополосный кадр пакуется до того, как кодируется верхний поддиапазон. Узкополосная часть битового потока представлена на таблице 9.1. Верхний поддиапазон расположен, как показано в таблице 11.1. Для широкополосного режима, ID режима такой же как и установка качества Speex, определенная в таблице 11.2. Это также означает, что широкополосный кадр может быть декодирован узкополосным декодировщиком только с одним предостережением, состоящем в том, что в одном пакете может быть более одного кадра, декодировщику будет необходимо пропускать широкополосные части для синхронизации битового потока.

Параметр	Частота обновления	0	1	2	3	4
Широкополосный бит	кадр	1	1	1	1	1
ID режима	кадр	3	3	3	3	3
ЛСП	кадр	0	12	12	12	12
Коэффициент усиления сигнала возбуждения	субкадр	0	5	4	4	4
Excitation VQ	субкадр	0	0	20	40	80
Итог	кадр	4	36	112	192	352

Table 11.1.: Расположение бит для верхнего диапазона в широкополосном режиме

Режим/качество	Битрейт (бит/с)	Качество/описание
0	3,950	Едва понятный (главным образом для комфортного шума)
1	5,750	Очень заметные артефакты/шум, плохая различимость
2	7,750	Очень заметные артефакты/шум, хорошая различимость
3	9,800	Артефакты/шум иногда надоедливы
4	12,800	Артефакты/шум обычно заметны
5	16,800	Артефакты/шум иногда заметны
6	20,600	Нужны хорошие наушники для того, чтобы заметить разницу
7	23,800	Нужны хорошие наушники для того, чтобы заметить разницу
8	27,800	Трудно заметить разницу даже в хороших наушниках
9	34,200	Трудно заметить разницу даже в хороших наушниках
10	42,200	Полностью прозрачен для голоса, хорошее качество музыки

Table 11.2.: Соотношение качества и битрейта для широкополосного кодировщика

A. Примеры кода

Данный раздел показывает пример кода для кодирования и декодирования речи с использованием Speex API. Команды, используемые для кодирование и декодирования файла могут быть следующими:
`% sampleenc in_file.sw | sampledec out_file.sw`
где оба файла сырые (без заголовка) закодированные при 16 битах на выборку (с естественным машинным порядком байт).

A.1. sampleenc.c

sampleenc принимает сырой файл с 16 битами на выборку, кодирует его и выдает поток Speex в stdout. Заметьте что используемый здесь формат файла **несовместим** с speexenc/speexdec.

Listing A.1: Source code for sampleenc

```
1  #include <speex/speex.h>
2  #include <stdio.h>
3
4  /*The frame size in hardcoded for this sample code but it doesn't have to be*/
5  #define FRAME_SIZE 160
6  int main(int argc, char **argv)
7  {
8     char *inFile;
9     FILE *fin;
10    short in[FRAME_SIZE];
11    float input[FRAME_SIZE];
12    char cbits[200];
13    int nbBytes;
14    /*Holds the state of the encoder*/
15    void *state;
16    /*Holds bits so they can be read and written to by the Speex routines*/
17    SpeexBits bits;
18    int i, tmp;
19
20    /*Create a new encoder state in narrowband mode*/
21    state = speex_encoder_init(&speex_nb_mode);
22
23    /*Set the quality to 8 (15 kbps)*/
24    tmp=8;
25    speex_encoder_ctl(state, SPEEX_SET_QUALITY, &tmp);
26
27    if (argc < 2)
28       return 1;
29
30    inFile = argv[1];
31    fin = fopen(inFile, "r");
32
33    /*Initialization of the structure that holds the bits*/
34    speex_bits_init(&bits);
35    while (1)
36    {
37       /*Read a 16 bits/sample audio frame*/
```

```
38         fread(in, sizeof(short), FRAME_SIZE, fin);
39         if (feof(fin))
40            break;
41         /*Copy the 16 bits values to float so Speex can work on them*/
42         for (i=0;i<FRAME_SIZE;i++)
43            input[i]=in[i];
44
45         /*Flush all the bits in the struct so we can encode a new frame*/
46         speex_bits_reset(&bits);
47
48         /*Encode the frame*/
49         speex_encode(state, input, &bits);
50         /*Copy the bits to an array of char that can be written*/
51         nbBytes = speex_bits_write(&bits, cbits, 200);
52
53         /*Write the size of the frame first. This is what sampledec expects but
54          it's likely to be different in your own application*/
55         fwrite(&nbBytes, sizeof(int), 1, stdout);
56         /*Write the compressed data*/
57         fwrite(cbits, 1, nbBytes, stdout);
58
59      }
60
61      /*Destroy the encoder state*/
62      speex_encoder_destroy(state);
63      /*Destroy the bit-packing struct*/
64      speex_bits_destroy(&bits);
65      fclose(fin);
66      return 0;
67   }
```

A.2. sampledec.c

sampledec считывает Speex поток из stdin, декодирует его, и выдает в файл с 16 битами на семпл. Заметьте что используемый здесь формат файла несовместим с speexenc/speexdec.

Listing A.2: Source code for sampledec

```
1  #include <speex/speex.h>
2  #include <stdio.h>
3
4  /*The frame size in hardcoded for this sample code but it doesn't have to be*/
5  #define FRAME_SIZE 160
6  int main(int argc, char **argv)
7  {
8     char *outFile;
9     FILE *fout;
10    /*Holds the audio that will be written to file (16 bits per sample)*/
11    short out[FRAME_SIZE];
12    /*Speex handle samples as float, so we need an array of floats*/
13    float output[FRAME_SIZE];
14    char cbits[200];
15    int nbBytes;
16    /*Holds the state of the decoder*/
17    void *state;
18    /*Holds bits so they can be read and written to by the Speex routines*/
19    SpeexBits bits;
```

```
20      int i, tmp;
21
22      /*Create a new decoder state in narrowband mode*/
23      state = speex_decoder_init(&speex_nb_mode);
24
25      /*Set the perceptual enhancement on*/
26      tmp=1;
27      speex_decoder_ctl(state, SPEEX_SET_ENH, &tmp);
28
29      outFile = argv[1];
30      fout = fopen(outFile, "w");
31
32      /*Initialization of the structure that holds the bits*/
33      speex_bits_init(&bits);
34      while (1)
35      {
36         /*Read the size encoded by sampleenc, this part will likely be
37           different in your application*/
38         fread(&nbBytes, sizeof(int), 1, stdin);
39         fprintf (stderr, "nbBytes: %d\n", nbBytes);
40         if (feof(stdin))
41            break;
42
43         /*Read the "packet" encoded by sampleenc*/
44         fread(cbits, 1, nbBytes, stdin);
45         /*Copy the data into the bit-stream struct*/
46         speex_bits_read_from(&bits, cbits, nbBytes);
47
48         /*Decode the data*/
49         speex_decode(state, &bits, output);
50
51         /*Copy from float to short (16 bits) for output*/
52         for (i=0;i<FRAME_SIZE;i++)
53            out[i]=output[i];
54
55         /*Write the decoded audio to file*/
56         fwrite(out, sizeof(short), FRAME_SIZE, fout);
57      }
58
59      /*Destroy the decoder state*/
60      speex_decoder_destroy(state);
61      /*Destroy the bit-stream truct*/
62      speex_bits_destroy(&bits);
63      fclose(fout);
64      return 0;
65  }
```

B. Jitter Buffer for Speex

Listing B.1: Example of using the jitter buffer for Speex packets

```
1  #include <speex/speex_jitter.h>
2  #include "speex_jitter_buffer.h"
3
4  #ifndef NULL
5  #define NULL 0
6  #endif
7
8
9  void speex_jitter_init(SpeexJitter *jitter, void *decoder, int sampling_rate)
10 {
11    jitter->dec = decoder;
12    speex_decoder_ctl(decoder, SPEEX_GET_FRAME_SIZE, &jitter->frame_size);
13
14    jitter->packets = jitter_buffer_init(jitter->frame_size);
15
16    speex_bits_init(&jitter->current_packet);
17    jitter->valid_bits = 0;
18
19 }
20
21 void speex_jitter_destroy(SpeexJitter *jitter)
22 {
23    jitter_buffer_destroy(jitter->packets);
24    speex_bits_destroy(&jitter->current_packet);
25 }
26
27 void speex_jitter_put(SpeexJitter *jitter, char *packet, int len, int timestamp)
28 {
29    JitterBufferPacket p;
30    p.data = packet;
31    p.len = len;
32    p.timestamp = timestamp;
33    p.span = jitter->frame_size;
34    jitter_buffer_put(jitter->packets, &p);
35 }
36
37 void speex_jitter_get(SpeexJitter *jitter, spx_int16_t *out, int *current_timestamp)
38 {
39    int i;
40    int ret;
41    spx_int32_t activity;
42    char data[2048];
43    JitterBufferPacket packet;
44    packet.data = data;
45    packet.len = 2048;
46
47    if (jitter->valid_bits)
48    {
```

```
49        /* Try decoding last received packet */
50        ret = speex_decode_int(jitter->dec, &jitter->current_packet, out);
51        if (ret == 0)
52        {
53           jitter_buffer_tick(jitter->packets);
54           return;
55        } else {
56           jitter->valid_bits = 0;
57        }
58     }
59
60     ret = jitter_buffer_get(jitter->packets, &packet, jitter->frame_size, NULL);
61
62     if (ret != JITTER_BUFFER_OK)
63     {
64        /* No packet found */
65
66        /*fprintf (stderr, "lost/late frame\n");*/
67        /*Packet is late or lost*/
68        speex_decode_int(jitter->dec, NULL, out);
69     } else {
70        speex_bits_read_from(&jitter->current_packet, packet.data, packet.len);
71        /* Decode packet */
72        ret = speex_decode_int(jitter->dec, &jitter->current_packet, out);
73        if (ret == 0)
74        {
75           jitter->valid_bits = 1;
76        } else {
77           /* Error while decoding */
78           for (i=0;i<jitter->frame_size;i++)
79              out[i]=0;
80        }
81     }
82     speex_decoder_ctl(jitter->dec, SPEEX_GET_ACTIVITY, &activity);
83     if (activity < 30)
84        jitter_buffer_update_delay(jitter->packets, &packet, NULL);
85     jitter_buffer_tick(jitter->packets);
86  }
87
88  int speex_jitter_get_pointer_timestamp(SpeexJitter *jitter)
89  {
90     return jitter_buffer_get_pointer_timestamp(jitter->packets);
91  }
```

C. IETF RTP Profile

AVT
Internet-Draft
Intended status: Standards Track
Expires: August 19, 2008

G. Herlein
J. Valin
CSIRO
A. Heggestad
Creytiv.com
A. Moizard
Antisip
February 16, 2008

RTP Payload Format for the Speex Codec
draft-ietf-avt-rtp-speex-05

Status of this Memo

Copyright Notice

Abstract

Speex is an open-source voice codec suitable for use in Voice over IP (VoIP) type applications. This document describes the payload format for Speex generated bit streams within an RTP packet. Also included here are the necessary details for the use of Speex with the Session Description Protocol (SDP).

Editors Note

All references to RFC XXXX are to be replaced by references to the RFC number of this memo, when published.

Table of Contents

1. Introduction

Speex is based on the CELP [CELP] encoding technique with support for either narrowband (nominal 8kHz), wideband (nominal 16kHz) or ultra-wideband (nominal 32kHz). The main characteristics can be summarized as follows:

- Free software/open-source
- Integration of wideband and narrowband in the same bit-stream
- Wide range of bit-rates available
- Dynamic bit-rate switching and variable bit-rate (VBR)
- Voice Activity Detection (VAD, integrated with VBR)
- Variable complexity

The Speex codec supports a wide range of bit-rates from 2.15 kbit/s to 44 kbit/s. In some cases however, it may not be possible for an implementation to include support for all rates (e.g. because of bandwidth, RAM or CPU constraints). In those cases, to be compliant with this specification, implementations MUST support at least narrowband (8 kHz) encoding and decoding at 8 kbit/s bit-rate (narrowband mode 3). Support for narrowband at 15 kbit/s (narrowband mode 5) is RECOMMENDED and support for wideband at 27.8 kbit/s (wideband mode 8) is also RECOMMENDED. The sampling rate MUST be 8, 16 or 32 kHz.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119 [RFC2119] and indicate requirement levels for compliant RTP implementations.

3. RTP usage for Speex

3.1. RTP Speex Header Fields

The RTP header is defined in the RTP specification [RFC3550]. This section defines how fields in the RTP header are used.

Payload Type (PT): The assignment of an RTP payload type for this packet format is outside the scope of this document; it is specified by the RTP profile under which this payload format is used, or signaled dynamically out-of-band (e.g., using SDP).

Marker (M) bit: The M bit is set to one on the first packet sent after a silence period, during which packets have not been transmitted contiguously.

Extension (X) bit: Defined by the RTP profile used.

Timestamp: A 32-bit word that corresponds to the sampling instant for the first frame in the RTP packet.

3.2. RTP payload format for Speex

The RTP payload for Speex has the format shown in Figure 1. No additional header fields specific to this payload format are required. For RTP based transportation of Speex encoded audio the standard RTP header [RFC3550] is followed by one or more payload data blocks. An optional padding terminator may also be used.

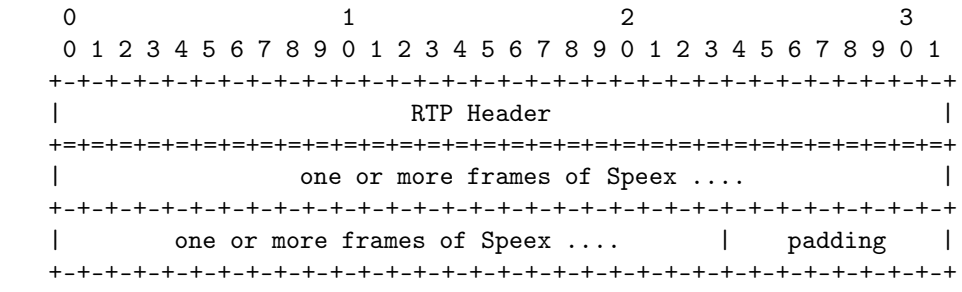


Figure 1: RTP payload for Speex

3.3. Speex payload

For the purposes of packetizing the bit stream in RTP, it is only necessary to consider the sequence of bits as output by the Speex encoder [speex_manual], and present the same sequence to the decoder. The payload format described here maintains this sequence.

A typical Speex frame, encoded at the maximum bitrate, is approx. 110 octets and the total number of Speex frames SHOULD be kept less than the path MTU to prevent fragmentation. Speex frames MUST NOT be fragmented across multiple RTP packets,

An RTP packet MAY contain Speex frames of the same bit rate or of varying bit rates, since the bit-rate for a frame is conveyed in band with the signal.

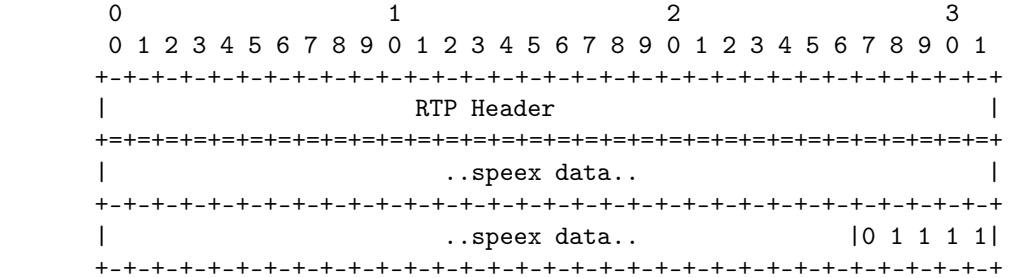
The encoding and decoding algorithm can change the bit rate at any 20 msec frame boundary, with the bit rate change notification provided in-band with the bit stream. Each frame contains both sampling rate (narrowband, wideband or ultra-wideband) and "mode" (bit-rate) information in the bit stream. No out-of-band notification is required for the decoder to process changes in the bit rate sent by the encoder.

The sampling rate MUST be either 8000 Hz, 16000 Hz, or 32000 Hz.

The RTP payload MUST be padded to provide an integer number of octets as the payload length. These padding bits are LSB aligned in network octet order and consist of a 0 followed by all ones (until the end of the octet). This padding is only required for the last frame in the packet, and only to ensure the packet contents ends on an octet boundary.

3.4. Example Speex packet

In the example below we have a single Speex frame with 5 bits of padding to ensure the packet size falls on an octet boundary.

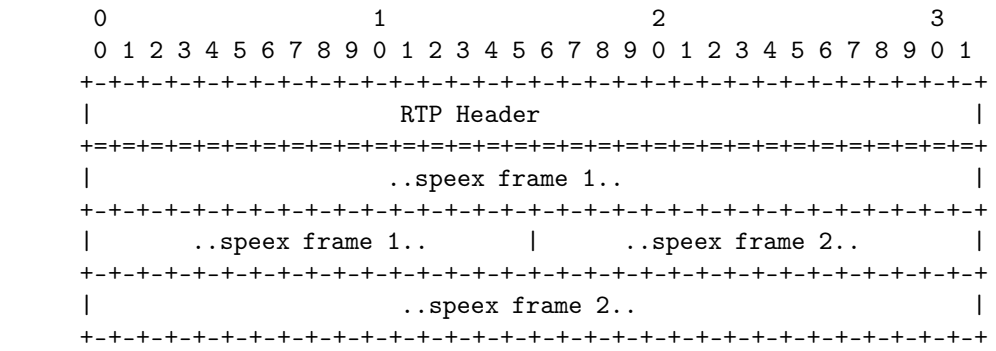


3.5. Multiple Speex frames in a RTP packet

Below is an example of two Speex frames contained within one RTP packet. The Speex frame length in this example fall on an octet boundary so there is no padding.

The Speex decoder [speex_manual] can detect the bitrate from the

payload and is responsible for detecting the 20 msec boundaries between each frame.



4. IANA Considerations

This document defines the Speex media type.

4.1. Media Type Registration

This section describes the media types and names associated with this payload format. The section registers the media types, as per RFC4288 [RFC4288]

4.1.1. Registration of media type audio/speex

Media type name: audio

Media subtype name: speex

Required parameters:

rate: RTP timestamp clock rate, which is equal to the sampling rate in Hz. The sampling rate MUST be either 8000, 16000, or 32000.

Optional parameters:

ptime: SHOULD be a multiple of 20 msec [RFC4566]

maxptime: SHOULD be a multiple of 20 msec [RFC4566]

vbr: variable bit rate - either 'on' 'off' or 'vad' (defaults to off). If on, variable bit rate is enabled. If off, disabled. If set to 'vad' then constant bit rate is used but silence will be encoded with special short frames to indicate a lack of voice for that period.

cng: comfort noise generation - either 'on' or 'off'. If off then silence frames will be silent; if 'on' then those frames will be filled with comfort noise.

mode: List supported Speex decoding modes. The valid modes are different for narrowband and wideband, and are defined as follows:

* {1,2,3,4,5,6,7,8,any} for narrowband

* {0,1,2,3,4,5,6,7,8,9,10,any} for wideband and ultra-wideband

Several 'mode' parameters can be provided. In this case, the

remote party SHOULD configure its encoder using the first supported mode provided. When 'any' is used, the offerer indicates that it supports all decoding modes. If the 'mode' parameter is not provided, the mode value is considered to be equivalent to 'mode=3;mode=any' in narrowband and 'mode=8;mode=any' in wideband and ultra-wideband. Note that each Speex frame does contains the mode (or bit-rate) that should be used to decode it. Thus application MUST be able to decode any Speex frame unless the SDP clearly specify that some modes are not supported. (e.g., by not including 'mode=any')

Encoding considerations:

This media type is framed and binary, see section 4.8 in [RFC4288].

Security considerations: See Section 6

Interoperability considerations:

None.

Published specification:

RFC XXXX [RFC Editor: please replace by the RFC number of this memo, when published]

Applications which use this media type:

Audio streaming and conferencing applications.

Additional information: none

Person and email address to contact for further information :

Alfred E. Heggestad: aeh@db.org

Intended usage: COMMON

Restrictions on usage:

This media type depends on RTP framing, and hence is only defined for transfer via RTP [RFC3550]. Transport within other framing protocols is not defined at this time.

Author: Alfred E. Heggestad

Change controller:

IETF Audio/Video Transport working group delegated from the IESG.

5. SDP usage of Speex

The information carried in the media type specification has a specific mapping to fields in the Session Description Protocol (SDP) [RFC4566], which is commonly used to describe RTP sessions. When SDP is used to specify sessions employing the Speex codec, the mapping is as follows:

- The media type ("audio") goes in SDP "m=" as the media name.
- The media subtype ("speex") goes in SDP "a=rtpmap" as the encoding name. The required parameter "rate" also goes in "a=rtpmap" as the clock rate.
- The parameters "ptime" and "maxptime" go in the SDP "a=ptime" and "a=maxptime" attributes, respectively.
- Any remaining parameters go in the SDP "a=fmtp" attribute by copying them directly from the media type string as a semicolon separated list of parameter=value pairs.

The tables below include the equivalence between modes and bitrates for narrowband, wideband and ultra-wideband. Also, the corresponding "Speex quality" setting (see SPEEX_SET_QUALITY in The Speex Codec Manual [speex_manual]) is included as an indication.

mode	Speex quality	bitrate
1	0	2.15 kbit/s
2	2	5.95 kbit/s
3	3 or 4	8.00 kbit/s
4	5 or 6	11.0 kbit/s
5	7 or 8	15.0 kbit/s
6	9	18.2 kbit/s
7	10	24.6 kbit/s
8	1	3.95 kbit/s

Mode vs Bitrate table for narrowband

Table 1

mode	Speex quality	wideband bitrate	ultra wideband bitrate
0	0	3.95 kbit/s	5.75 kbit/s
1	1	5.75 kbit/s	7.55 kbit/s
2	2	7.75 kbit/s	9.55 kbit/s
3	3	9.80 kbit/s	11.6 kbit/s
4	4	12.8 kbit/s	14.6 kbit/s
5	5	16.8 kbit/s	18.6 kbit/s
6	6	20.6 kbit/s	22.4 kbit/s
7	7	23.8 kbit/s	25.6 kbit/s
8	8	27.8 kbit/s	29.6 kbit/s
9	9	34.2 kbit/s	36.0 kbit/s
10	10	42.2 kbit/s	44.0 kbit/s

Mode vs Bitrate table for wideband and ultra-wideband

Table 2

The Speex parameters indicate the decoding capabilities of the agent, and what the agent prefers to receive.

The Speex parameters in an SDP Offer/Answer exchange are completely orthogonal, and there is no relationship between the SDP Offer and the Answer.

Several Speex specific parameters can be given in a single a=fmtp line provided that they are separated by a semi-colon:

```
a=fmtp:97 mode=1;mode=any;vbr=on
```

Some example SDP session descriptions utilizing Speex encodings follow.

5.1. Example supporting all modes, prefer mode 4

The offerer indicates that it wishes to receive a Speex stream at 8000Hz, and wishes to receive Speex 'mode 4'. It is important to understand that any other mode might still be sent by remote party: the device might have bandwidth limitation or might only be able to send 'mode=3'. Thus, application that support all decoding modes SHOULD include 'mode=any' as shown in the example below:

```
m=audio 8088 RTP/AVP 97
a=rtpmap:97 speex/8000
a=fmtp:97 mode=4;mode=any
```

5.2. Example supporting only mode 3 and 5

The offerer indicates the mode he wishes to receive (Speex 'mode 3'). This offer indicates mode 3 and mode 5 are supported and that no other modes are supported. The remote party MUST NOT configure its encoder using another Speex mode.

```
m=audio 8088 RTP/AVP 97
a=rtmap:97 speex/8000
a=fmtp:97 mode=3;mode=5
```

5.3. Example with Variable Bit Rate and Comfort Noise

The offerer indicates that it wishes to receive variable bit rate frames with comfort noise:

```
m=audio 8088 RTP/AVP 97
a=rtmap:97 speex/8000
a=fmtp:97 vbr=on;cng=on
```

5.4. Example with Voice Activity Detection

The offerer indicates that it wishes to use silence suppression. In this case vbr=vad parameter will be used:

```
m=audio 8088 RTP/AVP 97
a=rtmap:97 speex/8000
a=fmtp:97 vbr=vad
```

5.5. Example with Multiple sampling rates

The offerer indicates that it wishes to receive Speex audio at 16000 Hz with mode 10 (42.2 kbit/s), alternatively Speex audio at 8000 Hz with mode 7 (24.6 kbit/s). The offerer supports decoding all modes.

```
m=audio 8088 RTP/AVP 97 98
a=rtmap:97 speex/16000
a=fmtp:97 mode=10;mode=any
a=rtmap:98 speex/8000
a=fmtp:98 mode=7;mode=any
```

5.6. Example with ptime and Multiple Speex frames

The "ptime" attribute is used to denote the packetization interval (ie, how many milliseconds of audio is encoded in a single RTP packet). Since Speex uses 20 msec frames, ptime values of multiples of 20 denote multiple Speex frames per packet. Values of ptime which are not multiples of 20 MUST be rounded up to the first multiple of 20 above the ptime value.

In the example below the ptime value is set to 40, indicating that there are 2 frames in each packet.

```
m=audio 8088 RTP/AVP 97
a=rtpmap:97 speex/8000
a=ptime:40
```

Note that the ptime parameter applies to all payloads listed in the media line and is not used as part of an a=fmtp directive.

Care must be taken when setting the value of ptime so that the RTP packet size does not exceed the path MTU.

5.7. Example with Complete Offer/Answer exchange

The offerer indicates that it wishes to receive Speex audio at 16000 Hz, alternatively Speex audio at 8000 Hz. The offerer does support ALL modes because no mode is specified.

```
m=audio 8088 RTP/AVP 97 98
a=rtmap:97 speex/16000
a=rtmap:98 speex/8000
```

The answerer indicates that it wishes to receive Speex audio at 8000 Hz, which is the only sampling rate it supports. The answerer does support ALL modes because no mode is specified.

```
m=audio 8088 RTP/AVP 99
a=rtmap:99 speex/8000
```

6. Implementation Guidelines

Implementations that supports Speex are responsible for correctly decoding incoming Speex frames.

Each Speex frame does contains all needed informations to decode itself. In particular, the 'mode' and 'ptime' values proposed in the SDP contents MUST NOT be used for decoding: those values are not needed to properly decode a RTP Speex stream.

7. Security Considerations

RTP packets using the payload format defined in this specification are subject to the security considerations discussed in the RTP specification [RFC3550], and any appropriate RTP profile. This implies that confidentiality of the media streams is achieved by encryption. Because the data compression used with this payload format is applied end-to-end, encryption may be performed after compression so there is no conflict between the two operations.

A potential denial-of-service threat exists for data encodings using compression techniques that have non-uniform receiver-end computational load. The attacker can inject pathological datagrams into the stream which are complex to decode and cause the receiver to be overloaded. However, this encoding does not exhibit any significant non-uniformity.

As with any IP-based protocol, in some circumstances a receiver may be overloaded simply by the receipt of too many packets, either desired or undesired. Network-layer authentication may be used to discard packets from undesired sources, but the processing cost of the authentication itself may be too high.

8. Acknowledgements

The authors would like to thank Equivalence Pty Ltd of Australia for their assistance in attempting to standardize the use of Speex in H.323 applications, and for implementing Speex in their open source OpenH323 stack. The authors would also like to thank Brian C. Wiles <brian@streamcomm.com> of StreamComm for his assistance in developing the proposed standard for Speex use in H.323 applications.

The authors would also like to thank the following members of the Speex and AVT communities for their input: Ross Finlayson, Federico Montesino Pouzols, Henning Schulzrinne, Magnus Westerlund, Colin Perkins, Ivo Emanuel Goncalves.

Thanks to former authors of this document; Simon Morlat, Roger Hardiman, Phil Kerr.

9. Copying conditions

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

10.2. Informative References

[CELP] "CELP, U.S. Federal Standard 1016.", National Technical Information Service (NTIS) website http://www.ntis.gov/.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[speex_manual] Valin, J., "The Speex Codec Manual", Speex website http://www.speex.org/docs/.

Authors' Addresses

Greg Herlein
2034 Filbert Street
San Francisco, California 94123
United States

Email: gherlein@herlein.com

Jean-Marc Valin
CSIRO
PO Box 76
Epping, NSW 1710
Australia

Email: jean-marc.valin@usherbrooke.ca

Alfred E. Heggestad
Creytiv.com
Biskop J. Nilssonsgt. 20a
Oslo 0659
Norway

Email: aeh@db.org

Aymeric Moizard
Antisip
4 Quai Perrache
Lyon 69002
France

Email: jack@atosc.org

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).

D. Speex License

E. GNU Free Documentation License

Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are

not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author

or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Index