XNA, C#, ASP.NET ...

пятница, июля 06, 2007

Нашел часть перевода Bittorrent спецификации

Спецификация протокола Bittorrent v1.0

Определение

BitTorrent (краткое описание на wiki.theory.org) — это peer-to-peer протокол передачи файлов разработанный Брэмом Коэном (англ. *Bram Cohen*). Вы можете посетить его сайт [1]. BitTorrent разрабатывался для облегчения обмена файлами между пользователями в сети.

Цель

Цель этой спецификации — подробное документирование спецификации протокола BitTorrent версии 1.0. Страница спецификации протокола у Брэма [2] описывает его только в общих чертах, без подробной детализации некоторых вещей. Надеемся, что этот документ станет **формальной** спецификацией, написанной в доступном виде, с однозначными терминами, и который может быть использован как основа для обсуждения и освоения в будущем.

Документ предназначен для использования и поддержания его BitTorrent-сообществом разработчиков. Приглашаем каждого внести вклад в него, с пониманием, что его содержимое предназначено для предоставления сведений о текущем протоколе, который уже используется в большом количестве существующих реализаций клиентов.

Для заявок и предложений, проследуйте, пожалуйста, на страницу почтовой рассылки.

Область описания

Этот документ относится к первой версии (т.е. версии 1.0) спецификации протокола BitTorrent. В настоящее время он касается структуры torrent-файла, peerwire-протокола (протокола связи между пирами), и спецификации HTTP/HTTPS протокола Трекера. Описания новых ревизий каждого протокола должны быть изложены на их собственных отдельных страницах, **а не здесь**.

См. также

- http://www.bittorrent.org/protocol.html — Официальная спецификация протокола.
- http://wiki.theory.org/BitTorrentWishList — Список пожеланий разработчикам и конечным пользователям.
- http://wiki.theory.org/BitTorrentTrackerExtensions — Описание различных расширений в протоколе Трекера, которые сейчас используются.

Соглашения

В этом документе есть несколько соглашений, используемых для представления информации в лаконичной и однозначной форме.

- *пир* (англ. *peer* соучастник) и *клиент* (англ. *client*): Здесь, *пир* — это любой BitTorrent-клиент, принимающий участие в раздаче. *Клиент* — это тоже пир, но в данном случае, так называется сам BitTorrent-клиент, запущенный на локальном компьютере. Читатели этой спецификации могут думать о себе, как о *клиенте*, который соединяется с множеством *пиров*.
- *кусок* (англ. *piece*) и *блок*(англ. *block*): Здесь, *кусок* имеет отношение к порции скаченных данных, описанной в файле метаданных, которая может быть проверена SHA1-хешем. *Блок* — порция данных, которую *клиент* запрашивает у *пира*. Два или более *блоков* образуют целый *кусок*, который может быть проверен.

Архив блога

▼ 2007 (3)
▼ Июль (3)
Алгоритм Octree - теория и практика на DirectX
XNA GUI
Нашел часть перевода Bittorrent спецификации

Обо мне

gregsparrow
Просмотреть профиль

Links

Russian XNA resource
Russian gamedev resource

- *стандарт де-факто*: Большие блоки текста, набранные *курсивом*, указывают на общепринятые правила в различных реализациях BitTorrent-клиентов, которые считаются стандартом де-факто.

Чтобы помочь другим найти недавние изменения, сделанные в этом документе, пожалуйста, сделайте надлежащим образом запись об этом в логе изменений (англ. *change log*) (последний раздел). Она должна содержать краткую (т.е. в одну строчку) запись по каждому важному изменению, сделанному вами в документе.

Bencoding

Bencoding (b-кодирование) — это способ описания и организации данных в кратком формате. Он поддерживает следующие типы: байтовые строки, целые числа, списки и словари.

байтовые строки

Байтовые строки кодируются следующим образом: *<длина строки представленная в десятичной системе счисления ASCII>:<строковые данные>*

Обратите внимание, что нет определённого начального и конечного ограничителя.

> **Пример**: *4:spam* представляет строку "spam"

целые числа

Целые числа кодируются следующим образом: ***i***<*целое число представленное в десятичной системе счисления ASCII*>***e***

Начальное **i** и завершающее **e** являются начальным и конечным ограничителями. Вы можете кодировать отрицательные числа, например ***i-3e***. Нелья ставить в начале числа ноль, например ***i04e***. Однако, запись ***i0e*** является правильной.

> **Пример**: ***i3e*** представляет число "3"

ПРИМЕЧАНИЕ: Максимальное число битов этого целого точно не установлено, но обработка его как знакового 64-битного целого числа обязательна для обработки "больших файлов" в .torrent, размером более чем 4ГБ.

списки

Списки кодируются следующим образом: ***l***<*bencoded-данные*>***e***

Начальное **l** и завершающее **e** являются начальным и конечным ограничителями. Списки могут содержать любые из bencoded-типов, включая целые числа, строки, словари, и другие списки.

> **Пример:** ***l****4:spam4:eggs****e*** представляет список из двух строк: ["spam", "eggs"]

словари

Словари кодируются следующим образом: ***d***<*bencoded-строка*><*bencoded-элемент*>***e***

Начальное **d** и завершающее **e** являются начальным и конечным ограничителями. Обратите внимание, что ключи должны быть bencoded-строками. Значения могут быть любыми bencoded-типамм, включая целые числа, строки, списки, и другие словари. Ключи должны быть строками и представлены в отсортированном виде. Сортировку следует делать с помощью бинарного, а не "естественного" (алфавитно-цифрового), сравнения.

> **Пример**: ***d****3:cow3:moo4:spam4:eggs****e*** представляет словарь { "cow" => "moo", "spam" => "eggs" }
>
> **Пример**: ***d****4:spaml1:a1:bee* представляет словарь { "spam" => ["a", "b"] }
>
> **Пример**: ***d****9:publisher3:bob18:publisher.location4:home17:publisher-webpage15:www.example.com****e***

Структура файла метаданных

Все данные в файле метаданных (англ. *metainfo file*) находятся в bencode-формате. Подробное описания bencode-формата приведено выше.

Содержимое файла метаданных (расширение файла — ".torrent") — это bencoded-словарь, который содержит перечисленные ниже ключи. Все строковых величины закодированы в UTF-8.

- **info**: Словарь, который описывает файл(ы) торрента. Есть две возможных формы: первая — для случая с 'одно-файловым' торрентом, без структуры директорий; вторая — для 'много-файлового' торрента (см. подробности ниже).

- **announce**: Announce-URL (англ. *announce* публиковать, объявлять, давать знать) трекера (строка).

- **announce-list**: (Опционально) Это расширение официальной спецификации (обратно-совместимое). Ключ используется для списка резервных трекеров. Полное описание можно найти здесь.

- **creation date**: (Опционально) Дата создания торрента, в стандартном формате UNIX-времени (целое число секунд начиная с 1-Янв-1970 00:00:00 по UTC).

- **comment**: (Опционально) Текстовый комментарий в свободной форме от автора (строка).

- **created by**: (Опционально) Имя и версия программы, которая использовалась для создания torrent-файла (строка).

Словарь Info

Этот раздел содержит поля, общие для обоих режимов: "одно-файлового" и "много-файлового".

- **piece length**: Размер каждого куска в байтах (целое).

- **pieces**: Строка, составленная объединением всех 20-байтовых значений (по одному на каждый кусок) SHA1-хешей (байтовая строка).

- **private**: (Опционально) Это поле является целым числом. Если оно установлено в значение "1", клиент ОБЯЗАН сообщать о своём присутствии и получать список пиров ТОЛЬКО с помощью трекеров, перечисленных в файле метаданных. Если поле установлено в "0" или вообще отсутствует, клиент может получать список пиров другими способами, например с помощью PEX обмен пирами, DHT. Здесь, "приватный" можно читать как "без внешних источников списка пиров".

    - *ПРИМЕЧАНИЕ:* Это описание является "заглушкой", некоторые с ним не согласны. Используйте его на свой страх и риск. http://www.azureuswiki.com/index.php/Secure_Torrents — определение из azureus-вики.

    - Кроме того, следует отметить что, если даже это поле и используется на практике, оно не является частью официальной спецификации.

Info в одно-файловом режиме

В случае **одно-файлового** режима, словарь **info** содержит следующую структуру:

- **name**: Имя файла. Оно является чисто рекомендательным. (строка)

- **length**: Размер файла в байтах (целое).

- **md5sum**: (Опционально) 32-символьная шестнадцатеричная строка соответствующая MD5-сумме файла. Она не используется в BitTorrent, но включается некоторыми программами для совместимости.

Info в много-файловом режиме

В случае **много-файлового** режима, словарь **info** содержит следующую структуру:

- **name**: Имя директории, в которой содержатся все файлы. Оно является чисто рекомендательным. (string)
- **files**: Список словарей, по одному на каждый файл. Каждый словарь в этом списке содержит следующие ключи:
  - **length**: Размер файла в байтах (целое).
  - **md5sum**: (Опционально) 32-символьная шестнадцатеричная строка соответствующая MD5-сумме файла. Она не используется в BitTorrent, но включается некоторыми программами для совместимости.
  - **path**: Список, содержащий один или более строковых элементов, объединение которых даёт путь и имя файла. Каждый элемент в списке соответствует либо имени директории, либо (в случае с последнем элементом) — имени файла. Например, файл "dir1/dir2/file.ext" должен состоять из трёх строковых элементов: "dir1", "dir2", и "file.ext". Он кодируется как bencoded-список строк вот так ***l4**:dir1**4**:dir2**8**:file.ext**e***

Примечания

- Ключ **piece length** устанавливает номинальный размер куска, который, как правило, кратен 2. Размер куска обычно выбирается исходя из общего количества файловых данных в торренте, учитывая то, что слишком большой размер куска неэффективен, а слишком маленький даёт в результате большой .torrent файл метаданных. Используйте простой здравый смысл при выборе наименьшего размера куска, который не сделает .torrent файл размером больше, чем 50-75КБ (чтобы облегчить его загрузку на сервер). Однако, сейчас, когда размеры хостинга и ширина канала не сильно ограничиваются, *лучше делать размер куска 512КБ или меньше,* по крайней мере для торрентов примерно до 8-10ГБ, даже если это приведёт к увеличению торрент-файла, для более эффективной раздачи файлов. Часто используемые размеры — 256КБ, 512КБ, и 1МБ. *Каждый кусок равен выбранному размеру, за исключением последнего, размер которого может быть не равен выбранному. Число кусков вычисляется так: 'округление_в_большую_сторону(общая_длина / размер_куска)'. Для вычисления границ кусков для много-файловом случая, файловые данные рассматриваются как один длинный непрерывный поток, составленный конкатенацией всех файлов в порядке их следования в списке 'files'. Число кусков и их границы определяются таким же образом, как и в случае одного файла. Куски могут перекрывать границы файлов.*
- Каждый кусок имеет соответствующий ему SHA1-хеш. Эти хеши соединяются для формирования значения ключа **pieces** словаря **info**, описанного выше . Обратите внимание, что это **не** список, а строка. Её длина должна быть кратна 20.

HTTP/HTTPS протокол Трекера

Трекер — это HTTP/HTTPS сервис, который отвечает на HTTP GET запросы. Запросы включают в себя метрику от клиентов, помогающая трекеру вести статистику для торрента. В ответах содержится список пиров, который помогает клиенту учавствовать в раздаче. Базовый URL запроса состоит из "announce URL", который определён в файле метаданных (.torrent), и параметров, которые добавляются к этому URL'у, при помощи стандартног CGI-метода (т.е. добавление '?' после announce-URL, за которым следует последовательности 'параметр=значение', разделённые символом '&').

Обратите внимание, что все бинарные данные в URL (особенно info_hash и peer_id) должны быть должным образом экранированы). Это означает, что любой байт, который находится вне множеств *0-9*, *a-z*, *A-Z*, и *$-_.+!*'()*, должен быть закодирован в формате "%nn", где nn — шестнадцатеричное значение байта. (Подробнее см. в RFC 1738).

Параметры, используемые в GET запросе клиент->трекер:

- **info_hash**: 20-байтовый SHA1-хеш *значения* ключа *info* файла метаданных. Обратите внимание, что *значение* является bencoded-словарём. Определение *info*

было описано ранее. *Примечание: Это строка всегда url-кодирована, в отличии от* peer_id*, который должен быть незакодированным.*

- **peer_id**: 20-байтовая строка, используется как уникальный идентификатор клиента, сгенерированный им же при запуске. Может быть любым значением, в том числе и бинарным. *На данный момент нет никаких рекомендаций по генерации этого идентификатора. Однако, справедливо предположить, что он должен быть уникальным для локальной машины. Таким образом, вероятно, следует включать в него такие вещи, как идентификатор процесса и, возможно, временную метку, записанную при запуске. Способы кодирования этого поля основными клиентами см. ниже в разделе peer_id.*

- **port**: Номер порта, который прослушивает клиент. Стандартные порты, которые зарезервированы для BitTorrent — 6881-6889. Клиент может использовать любой другой порт, если он не может открыть его в указанном диапазоне.

- **uploaded**: Суммарное количество отданных данных (после того, как клиент послал событие 'started' трекеру) в десятичной системе счисления ASCII. Пока это точно не определено в официальной спецификации, считается, что здесь должно быть общее число отданных байт.

- **downloaded**: Суммарное количество скачанных данных (после того, как клиент послал событие 'started' трекеру) в десятичной системе счисления ASCII. Пока это точно не определено в официальной спецификации, считается, что здесь должно быть общее число скаченных байт.

- **left**: Число байт в десятичной системе счисления ASCII, которое клиент ещё должен скачать.

- **compact**: Служит признаком того, что клиент принимает компактные ответы. Список пиров заменяется строкой пиров, по 6 байт на один пир. Первые четыре байта — хост (в сетевом порядке байтов (англ. *network byte order*)), последние два байта — порт (опять же, в сетевом порядке байтов). Следует помнить, что некоторые трекеры поддерживают только компактные ответы (для экономии трафика) и игнорируют нормальные запросы.

- **event**: Если определено, значение должно быть одно из *started*, *completed*, *stopped*, либо пустое, что равнозначно неопределённому. Если точно не указано, то этот запрос выполняется через регулярные интервалы времени.

  - **started**: Первый запрос к трекеру *обязательно* должен включать в себя параметр *event* с этим значением.

  - **stopped**: Должно быть послано трекеру, если клиент правильно завершает работу.

  - **completed**: Должно быть послано трекеру при завершении закачки. Однако, это событие не должно посылаться, если при запуске клиента закачка уже на 100% завершена. По-видимому, это нужно для того, чтобы дать возможность трекеру правильно увеличивать показатель "завершённых закачек", который зависит от этого события.

- **ip**: (Опционально) Реальный IP-адрес клиентской машины, в формате адреса, состоящим из четырех частей, разделённых точками (англ. *dotted quad format*), или в формате шестнадцатеричного IPv6-адреса, определённом в RFC 3513. *Примечание: Вообще, этот параметр не является необходимым, так как адрес клиента может быть взят из IP-адреса, с которого отправлен запрос. Параметр нужен только в случае, когда IP-адрес, с которого пришёл запрос, не является IP-адресом клиента. Это происходит, когда клиент соединяется с трекером через прокси. А также, это необходимо, когда клиент и трекер находятся в одной локальной части NAT шлюза, т.к. иначе трекер будет выдавать внутренний (RFC 1918) адрес клиента, который не является маршрутизируемым. Поэтому, клиент должен однозначно установить IP-адрес (внешний, маршрутизируемый), для выдачи его внешним пирам. Разные трекеры обрабатывают этот параметр по-разному. Некоторые принимают его, если IP-адрес, с которого пришёл запрос, находится в диапазоне RFC 1918, другие — принимают безоговорочно, третьи — полностью его игнорируют. Случай с IPv6-адресом (например, 2001:db8:1:2::100) указывает на то, что клиент может общаться только по протоколу IPv6.*

- **numwant**: (Опционально) Число пиров, которое клиент хочет получить от трекера. Его значение может быть нулём. Если параметр не задан, обычно отдаётся 50 пиров.

- **key**: (Опционально) Дополнительная идентификация, которая не доступна остальным пользователям. Предназначена для того, чтобы дать клиенту подтвердить свою подлинность при смене IP-адреса.

- **trackerid**: (Опционально) Если предыдущее оповещение (англ. *announce*) содержит идентификатор трекера, он должен быть установлен здесь.

Трекер отвечает "text/plain" документом, содержащим bencoded-словарь со следующими ключами:

- **failure reason**: Если задан, то никаких других ключей в документе нет. Его значением является сообщение об ошибке, в читабельном виде, говорящее о том, почему запрос неудался (строка).

- **warning message**: (Новый) Похож на *failure reason*, но ответ всё же нормально обрабатывается. Это предупреждающее сообщение показывается также, как и ошибка.

- **interval**: Интервал в секундах, который клиент должен выдерживать между посылкой регулярных запросов трекеру (принудительный).

- **min interval**: Минимальный интервал для оповещений. Если задан, клиент не должен делать оповещения чаще, чем это значение.

- **tracker id**: Строка, которую клиент должен посылать обратно в его последующих оповещениях. Если отсутствует или предыдущее оповещение содержало идентификатор трекера, не нужно сбрасывать старое значание — используйте его.

- **complete**: Число пиров с полным файлом, то есть *сидов* (англ. *seed* зерно, источник) (целое).

- **incomplete**: Число не-сидов, так называемых *личеров* (англ. *leech* пиявка) (целое).

- **peers**: Значением этого ключа является список словарей, каждый из которых включает следующие ключи:

    - **peer id**: Выбранный самим пиром идентификатор для запросов трекеру, описанный выше (строка).

    - **ip**: IP-адрес пира (либо IPv6, либо IPv4) или DNS-имя (строка).

    - **port**: Номер порта пира (целое).

Как упоминалось ранее, список пиров, по-умолчанию, имеет 50 записей. Если торрент имеет небольшое количество пиров, список будет меньше. В противном случае, трекер выбирает пиры для списка случайным образом. *Трекер может использовать более интеллектуальный алгоритм для осуществления выборки пиров для ответа на запрос. Например, не сообщать о сидах другим сидам.*

Клинты могут посылать запросы трекеру чаще, чем с заданным интервалом, если произошло какое-либо событие (например, задержка или завершение закачки), либо клиенту нужно узнать о большем количестве пиров. Тем не менее, считается плохой практикой "долбёжка" (англ. *hammer* бить, наносить удары) трекера для получения списков пиров. Если клиент хочет получить список большего размера в ответе, он должен использовать параметр **numwant** в запросе.

***Примечание разработчика**: Даже 30 пиров **достаточно**. Фактически, официальный клиент 3-ей версии активно создаёт новые соединения, только если имеет менее 30 пиров, и отказывает в соединении, при более чем 55 пирах. **Это значение важно для производительности.** Когда новый кусок полностью скачен, большинству активных пиров должно быть послано HAVE-сообщение (см. ниже). В результате, количество трафика увеличивается пропорционально количеству пиров. При их количестве большем, чем 25, весьма маловероятно, что новые пиры поднимут скорость скачивания.*

*Разработчикам клиентов **настоятельно** рекомендуется сделать этот параметр незаметным и сложным для изменения, т.к. только в редких случаях он будет полезен.*

Соглашение о 'scrape'

По договорённости большинство трекеров поддерживают другую форму запроса, для получения информации о состоянии получаемого торрента (или всех торрентов), которым управляет трекер. Он обращается к так называемой "scrape-странице" (англ. *scrape* царапанье, скобление), т.к. это автоматизирует утомительный процесс "scraping'a экрана" страницы статистики трекера.

При составлении scrape-URL используется такой же HTTP GET метод, подобный описанному выше. Только базовый URL другой. Для получения scrape-URL'a проделываются следующие шаги. Начинаем с announce-URL. Находим в нём последний символ '/'. Если текст непосредственно следующий за '/' не 'announce', это признак того, что трекер не поддерживает scrape. В противном случае, заменяем 'announce' на 'scrape' для нахождения scrape-страницы.

Примеры: (announce-URL -> scrape-URL)

```
  ~http://example.com/announce           -> ~http://example.com/scrape
 ~http://example.com/x/announce         -> ~http://example.com/x/scrape
 ~http://example.com/announce.php       -> ~http://example.com/scrape.php
 ~http://example.com/a                  -> (scrape не поддерживается)
 ~http://example.com/announce?x<code>2%0644 -> ~http://example.com/scrape?x</code
>2%0644
 ~http://example.com/announce?x=2/4     -> (scrape не поддерживается)
 ~http://example.com/x%064announce      -> (scrape не поддерживается)
```

Обратите особое внимание, that entity unquoting is *not* to be done. Этот стандарт документирован Брэмом в архиве списка разработки BitTorrent: http://groups.yahoo.com/group/BitTorrent/message/3275

Scrape-URL может быть дополнен опциональным параметром *info_hash*, 20-байтовое значение, которое было описано выше. Это ограничит отчёт трекера только этим одиночным торрентом. В противном случае, будет отдаваться статистика для всех торрентов, которыми управляет трекер. Авторам программ сильно рекомендуется использование параметра *info_hash*, если это возможно, для уменьшения нагрузки на трекера и канал.

Ответом на этот HTTP GET запрос является "text/plain" документ, содержащий bencoded-словарь со следующими ключами:

- **files**: Словарь, содержащий одну пару ключ/значение для каждого торрента, для которых есть статистика. Если даётся правильный *info_hash*, то словарь содержит одну пару ключ/значение. Каждый ключ состоит из 20-байтового бинарного значения info_hash. Значение ключа — это ещё один словарь:
  - **complete**: Число пиров с полным файлом, то есть *сидов* (целое).
  - **downloaded**: Общее количество завершённых закачек, которое зарегистрировал трекер ("event=complete", то есть клиент закончил скачивание торрента).
  - **incomplete**: Число не-сидов, так называемых *личеров* (целое).
  - **name**: (Опционально) Внутреннее имя торрента, указанное в ключе *name* раздела *info* .torrent файла.

Обратите внимание, что этот ответ имеет три уровня вложенных словарей. Вот пример:

```
d5:filesd20:....................d8:completei5e10:downloadedi50e10:incompletei10ee
ee
```

Где `....................` это 20 байт info_hash, и имеется 5 сидов, 10 личеров и 50 завершённых закачек.

Неофициальные расширения к scrape

Ниже приведены ключи, которые используются в ответе, но не являются официальными. Пока они являются неофициальным, эти ключи необязательны.

- **failure reason**: Сообщение об ошибке, в читабельном виде, говорящее о том, почему запрос неудался (строка). Клиенты, обрабатывающие этот ключ: Azureus.

- **flags**: Словарь, содержащий разнообразные флаги. Значения этих флагов это ещё один вложенный словарь, который может содержать:

    - **min_request_interval**: Значение этого ключа — целое число, которое определяет, сколько секунд клиент должен ждать перед отправкой следующег scrape-запроса трекеру. Трекеры, посылающие этот ключ: BNBT. Клиенты, которые его обрабатывают: Azureus.

Протокол связи между пирами (TCP)

Протокол связи между пирами (англ. *peer wire protocol*), далее именуется как *peerwire-протокол* или *peer-протокол*.

Обзор

Peer-протокол помогает обмену кусками, описанных в файле **метаданных**.

Здесь обратите внимание, что в оригинальной спецификации используется термин "кусок" при описания peer-протокола, но это не тот термин "кусок", который используется в описании файла метаданных. По этой причине, в этой спецификации будет использоваться термин "блок" для описания данных, которыми обмениваются пиры по сети.

Клиент должен поддерживать информацию о состоянии каждого соединения, которое он имеет с удалённым пиром:

- **choked**: Блокирует ли (англ. *choke* душить, пережимать) удалённый пир этого клиента или нет. Если пир блокирует клиента, это означает, что пир не будет отвечать на любой запрос клиента до тех пор, пока не разблокирует его. Клиенту не следует пытаться запрашивать блоки, т.к. все ожидающие ответа (неотвеченные) запросы будут отброшены пиром.

- **interested**: Заинтересован ли удалённый пир в чём-то, что может предложить клиент. Это означает, что удалённый пир начнёт запрашивать блоки, когда клиент разблокирует его.

Обратите внимание, что это также означает, что клиент тоже будет следить за тем, заинтересован ли он в удалённом пире или нет, и заблокирован ли он удалённым пиром или нет. Поэтому, реальный список выглядит примерно так:

- **am_choking**: этот клиент блокирует пира

- **am_interested**: этот клиент заинтересован в пире

- **peer_choking**: пир блокирует этого клиента

- **peer_interested**: пир заинтересован в этом клиенте

Клиент начинает соединие как "заблокированный" и "не заинтересованный". Другими словами:

- **am_choking** = 1

- **am_interested** = 0

- **peer_choking** = 1

- **peer_interested** = 0

Блок скачивается клиентом тогда, когда он заинтересован в пире, и пир не блокирует клиента. Блок отдаётся клиентом тогда, когда он не блокирует пира, и пир заинтересован в клиенте.

Для клиента важно информировать пиры о том, заинтересован ли он а них или нет. Информацию об этом состоянии следует своевременно поддерживать с каждом пиром, даже если клиент им заблокирован. Это позволяет пирам знать, начнёт ли клиент скачивание, когда он его разблокирует (и наоборот).

Типы данных

Если не указан другой способ, все целые числа в peerwire-протоколе кодируются как четырёх байтовые значения big-endian формата, включая префиксный размер у всех сообщений, которые приходят после установки связи.

Поток сообщений

Peerwire-протокол состоит из начальной установки связи (см. ниже) и последующего обмена сообщениями с префикс-размером (англ. *length-prefixed messages*) между пирами. Префикс-размер — это целое число, описанное выше.

"Рукопожатие"

"Рукопожатие" (англ. *handshake*), далее именуется как *хендшейк* для обозначения сообщения, используемого для установления связи. Процесс подтверждение установления связи (англ. *handshaking*), далее именуется как *установление связи*.

Хендшейк — это обязательное сообщение и оно должно быть первым сообщением, переданное клиентом.

handshake: <pstrlen><pstr><reserved><info_hash><peer_id>

- **pstrlen**: Длина <pstr> строки, один байт.
- **pstr**: Строка идентификатора протокола.
- **reserved**: Восемь (8) зарезервированных байт. Все текущие реализации заполняют их нулями. Каждый бит в этих байтах может использоваться для изменения режима работы протокола. *В своём email'е Брэм предлагает в первую очередь использовать младшие биты для того, чтобы старшие биты применялись для изменения значения младших.*
- **info_hash**: 20-байтовый SHA1-хеш ключа info файла метаданных. Это тот же info_hash, который передаётся в запросах трекеру.
- **peer_id**: 20-байтовая строка, используется как уникальный идентификатор клиента. Это тот же peer_id, который передаётся в запросах трекеру.

В версии 1.0 протокола BitTorrent, pstrlen = 19, и pstr = "BitTorrent protocol".

Инициатор соединения ожидает немедленной пересылки его хендшейка. Адресат может ожидать хендшейк от инициатора, если он обслуживает несколько торрентов одновременно (торренты однозначно идентифицируются по их info_hash). Несмотря на это, адресат должен ответить сразу, как только он увидит поле info_hash в хендшейке. Трекерная функция NAT-проверки не посылает поле peer_id в хендшейке.

Если клиент получает хендшейк с info_hash, который он ещё не обслуживает, то он должен оборвать соединение.

Если инициатор соединения получает хендшейк, в котором peer_id не совпадает с ожидаемым peer_id, то инициатор закрывает соединение. *Обратите внимание, что инициатор, по-видимому, получает информацию о пире от трекера, которая включает в себя peer_id этого пира. Поэтому, peer_id от трекера и peer_id в хендшейке должны совпадать.*

peer_id

Есть два основных соглашения кодирования информации о клиенте и его версии в peer_id: Azareus-стиль и Shadow's-стиль.

Azareus-стиль использует следующее кодирование: '-', два символа для идентификатора клиента, четыре ascii цифры для номера версии, '-', далее случайные числа.

Например: '-AZ2060-'...

Известные клиенты, которые используют этот стиль кодирования:

- 'AR' - Arctic
- 'AX' - BitPump

- 'AZ' - Azureus
- 'BB' - BitBuddy
- 'BC' - BitComet
- 'BF' - Bitflu
- 'BR' - BitRocket
- 'BS' - BTSlave
- 'BX' - ~Bittorrent X
- 'CD' - Enhanced CTorrent
- 'CT' - CTorrent
- 'DE' - DelugeTorrent
- 'EB' - EBit
- 'ES' - electric sheep
- 'HL' - Halite
- 'KT' - KTorrent
- 'LP' - Lphant
- 'LT' - libtorrent
- 'lt' - libTorrent
- 'MP' - MooPolice
- 'MT' - MoonlightTorrent
- 'qB' - qBittorrent
- 'QT' - Qt 4 Torrent example
- 'RT' - Retriever
- 'SB' - ~Swiftbit
- 'SS' - SwarmScope
- 'SZ' - Shareaza
- 'TN' - TorrentDotNET
- 'TR' - Transmission
- 'TS' - Torrentstorm
- 'UL' - uLeecher!
- 'UT' - µTorrent

- 'XT' - XanTorrent
- 'ZT' - ZipTorrent

Shadow's-стиль использует следующее кодирование: один алфавитно-цифровой ascii символ для идентификатора клиента, три ascii цифры для номера версии , '——', далее случайные числа.

Например: 'S587——'...

Известные клиенты, которые используют этот стиль кодирования:

- 'A' - ABC
- 'O' - Osprey Permaseed
- 'R' - Tribler
- 'S' - Shadow's client
- 'T' - BitTornado
- 'U' - UPnP NAT Bit Torrent

Клиент Брэма сейчас использует такой стиль... 'M3-4-2—' или 'M4-20-8-'.

BitComet использует несколько другой стиль. Его peer_id состоит из четырёх ascii символов 'exbc', двух байт x и y, далее случайные числа. Номер версии x — десятичная цифра до десятичной точки, и y — две десятичных цифры после точки. BitLord использует аналогичную схему, но добавляет 'LORD' после номера версии. неофициальный патч для BitComet заменяет 'exbc' на 'FUTB'. Кодирование peer_id в BitComet измененилось на Azureus-стиль с версии BitComet 0.59.

XBT Client также имеет свой стиль. Его peer_id состоит из трёх символов 'XBT' в верхнем регистре, плюс три ascii цифры? представляющие номер версии. Если это отладочный билд (англ. *debug build*) клиента, седьмой байт — это символ 'd' в нижнем регистре, иначе он равен '-'. Далее следует символ '-' и случайные цифры, и символы в верхнем и нижнем регистре. Например: 'XBT054d-' в начале, указывают на отладочный билд версии 0.5.4.

Opera 8 previews используют следующую схему для peer_id: Первые два символа 'OP', и четыре цифры равные номеру билда. Все последующие символы — случайные шестнадцатеричные символы нижнего регистра.

MLdonkey использует такую схему: Первые символы '-ML', далее версия, разделённая точками (англ. *dotted version*), потом символ '-' и случайная последовательность. Например '-ML2.7.2-kgjjfkd'

Bits on Wheels использует шаблон '-BOWAxx-yyyyyyyyyyyy', где y — случайные символы верхнего регистра, а x зависит от версии. Версия 1.0.6 имеет xx = 0C.

Queen Bee использует новый стиль Брэма: 'Q1-0-0—' или 'Q1-10-0-' с последующими случайными байтами.

Многие клиенты используют полностью случайные номера или 12 нулей со следующими случайными числами (как старые версии клиента Брэма).

Сообщения

Все оставшиеся сообщения протокола имеют такой вид

<префикс-длина><идентификатор сообщения><тело сообщения>.

Префикс-длина — это четырёх байтовое big-endian значение. Идентификатор сообщения — одиночная десятичная цифра.

keep-alive: <len=0000>

keep-alive сообщение — это пустое сообщение, с префикс-длиной равной нулю. В нём нет ни идентификатора, ни тела сообщения. Пиры могут закрыть соединение, если они не получают

keep-alive или любое другое сообщение за определённый период времени. Поэтому, оно должно быть послано для того, чтобы сохранить соединение, если в установленное количество времени (оно обычно равно двум минутам) нет определённых комманд для посылки пиру.

choke: <len=0001><id=0>

choke сообщение — с фиксированой длиной и без тела сообщения.

unchoke: <len=0001><id=1>

unchoke сообщение — с фиксированой длиной и без тела сообщения.

interested: <len=0001><id=2>

interested сообщение — с фиксированой длиной и без тела сообщения.

not interested: <len=0001><id=3>

not interested сообщение — с фиксированой длиной и без тела сообщения.

have: <len=0005><id=4><piece index>

have сообщение с фиксированой длиной. Тело сообщения — индекс, с отсчётом от нуля, (англ. *zero-based index*) куска, который только что был упешно закачен и проверен с помощью хеша.

Примечание разработчика: Это строгое определение, но в реальности некоторые могут хитрить. В частности потому, что крайне маловероятно, что пиры скачают куски, которые у них уже есть; пир, который имеет у себя кусок, может не известить об этом другого пира, также имеющего этот кусок. "Сокрытие HAVE" приводит как минимум к 50% сокращению числа HAVE сообщений, что приводит к приблизительно 25-35% снижению накладных расходов протокола (англ. protocol overhead*). Но в тоже время, имеет смысл посылать HAVE сообщение пиру, уже имеющего этот кусок, т.к. это полезно для определения редких кусков.*

*Злой пир может также сообщать о наличии у него только тех кусков, который пир качать не будет. Поэтому, попытка моделировать информацию о пирах используя эту информацию — **плохая идея**.*

bitfield: <len=0001+X><id=5><bitfield>

bitfield сообщение может быть послано только сразу же после того, как завершено установление связи, и до посылки других сообщений. Оно необязательное, и нет необходимости посылать его, если клиент не имеет кусков.

bitfield сообщение имеет переменную длину, где X — длина поля bitfield. Тело сообщения — это битовое поле показывающее, какие куски успешно скачены. Старший бит в первом байте соответствует куску с индексом 0. Нулевой бит указывает на отсутствующий кусок, установленный бит — на проверенный и доступный кусок. Лишние биты на конце устанавливаются в ноль.

bitfield неверной длины рассматривается как ошибка. Клиентам следует обрывать соединение, если они получают bitfield, имеющий неправильный размер или несколько лишних наборов бит.

request: <len=0013><id=6><index><begin><length>

request сообщение фиксированной длины, используется для запроса блоков. Тело сообщения включает в себя следующую информацию:

- **index**: целое, указывающее на индекс (с отсчётом от нуля) куска
- **begin**: целое, указывающее на смещение (с отсчётом от нуля) внутри куска
- **length**: целое, указывающее на размер запрошенного блока.

Точка зрения #1 Согласно официальной спецификации, "Все текущие реализации используют размер блока равный 2^15 (32КБ), и закрывают соединения, которые запрашивают больше, чем 2^17 (128КБ)." Начиная с версии 3 или 2004г., этот размер изменился на 2^14 (16КБ). В версии 4.0 или середина-2005г., официальный клиент разрывает связь при запросах более чем 2^14 (16КБ); и некоторые клиенты делают также. Обратите внимание, что размеры запрошенных

блоков меньше, чем размер куска (>=2^18 байт), поэтому требуется несколько запросов, чтобы скачать полный кусок.

Определённо, спецификация разрешает использовать 2^15 (32КБ) запросы. Но в реальности, почти все клиенты используют 2^14 (16КБ) запросы. В следствие того, что клиенты используют этот размер, рекомендуется делать его именно таким. Из-за меньших запросов, приводящих к увеличению накладных расходов в следствие большего числа запросов, разработчики отказываются от использования размера меньше 2^14 (16КБ).

Выбор принудительного ограничения для размера запрошенного блока не так прост. Т.к. официальный клиент версии 4 вынуждает использовать 16КБ запросы, большинство клиентов используют именно этот размер. В то же время, сейчас 2^14 (16КБ) является полу-*официальным ограничением (только* полу, *т.к. официальный документ по протолу не был обновлён), поэтому такое ограничение неправильно. Одновременно с этим, разрешение больших запросов увеличивает набор возможных пиров, и только за исключением очень узких каналов (<256kbps), несколько блоков будут скачиваться за один период разблокировки-блокировки (англ.* choke-timeperiod*). Таким образом, простое принуждение к использованию старого лимита служит причиной минимального ухудшения производительности. Благодаря этому фактору, рекомендуется ограничивать его в пределах старого 2^17 (128КБ) максимального размера.*

Точка зрения #2 Этот раздел содержал ложную информацию большую часть времени существования страницы. Это третий раз, когда я исправляю одну и ту же часть после добавления в неё неверной информации, поэтому я не хочу переписывать его полностью, т.к. возможно он будет опять переделан... В текущей версии есть по меньшей мере следующие ошибки: Официальный клиент использовал 2^14 (16384) байтовые запросы, до тех пор, пока он был единственным существующим клиентом; только "официальная спецификация" все ещё говорит об устарелом 32768 байтовом значении, которое в реальности ни является размером по-умолчанию, ни максимально разрешённым. В версии 4, характеристика запроса не изменилась, но максимальный разрешённый размер изменился на размер по-умолчанию. В последних версиях официального клиента, максимальный размер изменился на 32768 (учтите, что это первое появление числа 32768 и для максимального, и для размера по-умолчанию, начиная с первой древней версии). То, что "большинство старых клиентов используют 32КБ запросы" — неправда. Discussion of larger requests fails to take latency effects into account.

piece: <len=0009+X><id=7><index><begin><block>

piece сообщение переменной длины, где X — длина блока. Тело сообщения включает в себя следующую информацию:

- **index**: целое указывающие на индекс (с отсчётом от нуля) куска
- **begin**: целое указывающие на смещение (с отсчётом от нуля) внутри куска
- **block**: блок данных, который является подмножеством куска, заданного индексом.

cancel: <len=0013><id=8><index><begin><length>

cancel сообщение фиксированной длины, используется для отмены запросов блоков. Тело сообщения идентично телу в *request*-сообщении. Обычно используется в процессе "Конца Игры" (см. ниже раздел Алгоритмы).

port: <len=0003><id=9><listen-port>

port сообщение посылается новыми версиями официального клиента, который снабжает DHT трекер. listen-port — это порт, который прослушивается DHT-узлом этого пира. Пир должен быть помещён в локальную таблицу маршрутизации (если есть поддержка DHT-трекера).

Алгоритмы

Организация очереди

View #1 In general peers are advised to keep a few unfullfilled requests on each connection. This is done because otherwise a full round trip is required from the download of one block to begining the download of a new block (round trip between PIECE message and next REQUEST message). On links with high BDP (bandwidth-delay-product, high latency or high bandwidth), this can result in a substantial performance loss.

Implementer's note: This the 'most crucial performance item. *A static queue of 10 requests is reasonable for 16KB blocks on a 5mbps link with 50ms latency. Links with greater bandwidth are becoming very common so UI designers are urged to make this readily available for changing.*

Notably cable modems were known for traffic policing and increasing this might of aleviated some of the problems caused by this.

View #2 NOTE: much of the information in this "Queuing" section is false or misleading. I'll just note that the "defaults to 5 outstanding requests" hasn't been true for a long time, "32 KB blocks" is misleading since you normally don't use 32 KB blocks, and tuning queue length by changing it and trying to measure the effects is a bad idea.

Super Seeding

(This was not part of the original specification)

The super-seed feature in S-5.5 and on is a new seeding algorithm designed to help a torrent initiator with limited bandwidth "pump up" a large torrent, reducing the amount of data it needs to upload in order to spawn new seeds in the torrent.

When a seeding client enters "super-seed mode", it will not act as a standard seed, but masquerades as a normal client with no data. As clients connect, it will then inform them that it received a piece — a piece that was never sent, or if all pieces were already sent, is very rare. This will induce the client to attempt to download only that piece.

When the client has finished downloading the piece, the seed will not inform it of any other pieces until it has seen the piece it had sent previously present on at least one other client. Until then, the client will not have access to any of the other pieces of the seed, and therefore will not waste the seed's bandwidth.

This method has resulted in much higher seeding efficiencies, by both inducing peers into taking only the rarest data, reducing the amount of redundant data sent, and limiting the amount of data sent to peers which do not contribute to the swarm. Prior to this, a seed might have to upload 150% to 200% of the total size of a torrent before other clients became seeds. However, a large torrent seeded with a single client running in super-seed mode was able to do so after only uploading 105% of the data. This is 150-200% more efficient than when using a standard seed.

Super-seed mode is NOT *recommended for general use. While it does assist in the wider distribution of rare data, because it limits the selection of pieces a client can downlad, it also limits the ability of those clients to download data for pieces they have already partially retrieved. Therefore, super-seed mode is only recommended for initial seeding servers.*

Why not rename it to e.g. "Initial Seeding Mode" or "Releaser Mode" then?

Piece downloading strategy

Clients may choose to download pieces in random order.

A better strategy is to download pieces in rarest first *order. The client can determine this by keeping the initial bitfield from each peer, and updating it with every* have *message. Then, the client can download the pieces that appear least frequently in these peer bitfields.*

When a download is almost complete, there's a tendency for the last few blocks to trickle in slowly. To speed this up, the client sends requests for all of its missing blocks to all of its peers. To keep this from becoming horribly inefficient, the client also sends a cancel to everyone else every time a block arrives.

There is no documented thresholds, recommended percentages, or block counts that could be used as a guide or Recommended Best Practice here.

When to enter end game mode is an area of discussion. Some clients enter end game when all pieces have been requested. Others wait until the number of blocks left is lower than the number of blocks in transit, and no more than 20. There seems to be agreement that it's a good idea to keep the number of pending blocks low (1 or 2 blocks) to minimize the overhead, and if you randomize the blocks requested, there's a lower chance of downloading duplicates. More on the protocol overhead can be found here: http://hal.inria.fr/inria-00000156/en

Choking and Optimistic Unchoking

Choking is done for several reasons. TCP congestion control behaves very poorly when sending over many connections at once. Also, choking lets each peer use a tit-for-tat-ish algorithm to ensure that they get a consistent download rate.

The choking algorithm described below is the currently deployed one. It is very important that all new algorithms work well both in a network consisting entirely of themselves and in a network consisting mostly of this one.

There are several criteria a good choking algorithm should meet. It should cap the number of simultaneous uploads for good TCP performance. It should avoid choking and unchoking quickly, known as 'fibrillation'. It should reciprocate to peers who let it download. Finally, it should try out unused connections once in a while to find out if they might be better than the currently used ones,

known as optimistic unchoking.

The currently deployed choking algorithm avoids fibrillation by only changing choked peers once every ten seconds.

Reciprocation and number of uploads capping is managed by unchoking the four peers which have the best upload rate and are interested. This maximizes the client's download rate. These four peers are referred to as *downloaders*, because they are interested in downloading from the client.

Peers which have a better upload rate (as compared to the *downloaders*) but aren't interested get unchoked. If they become interested, the *downloader* with the worst upload rate gets choked. If a client has a complete file, it uses its upload rate rather than its download rate to decide which peers to unchoke.

For optimistic unchoking, at any one time there is a single peer which is unchoked regardless of it's upload rate (if interested, it counts as one of the four allowed *downloaders*). Which peer is optimistically unchoked rotates every 30 seconds. Newly connected peers are three times as likely to start as the current optimistic unchoke as anywhere else in the rotation. This gives them a decent chance of getting a complete piece to upload.

Anti-snubbing

Occasionally a BitTorrent peer will be choked by all peers which it was formerly downloading from. In such cases it will usually continue to get poor download rates until the optimistic unchoke finds better peers. To mitigate this problem, when over a minute goes by without getting a single piece from a particular peer, BitTorrent assumes it is "snubbed" by that peer and doesn't upload to it except as an optimistic unchoke. This frequently results in more than one concurrent optimistic unchoke, (an exception to the exactly one optimistic unchoke rule mentioned above), which causes download rates to recover much more quickly when they falter.

Официальные расширения к протоколу

Currently there are a few official extensions to the protocol.

Fast Peers Extensions

- Reserved Bit: The third least significant bit in the 8th reserved byte i.e. reserved[7] = 0x04

These extensions serve multiple purposes. They allow a peer to more quickly bootstrap into a swarm by giving a peer a specific set of pieces which they will be allowed download regardless of choked status. They reduce message overhead by adding HaveAll and HaveNone messages and allow explicit rejection of piece requests whereas previously only implicit rejection was possible meaning that a peer might be left waiting for a piece that would never be delivered.

The specificication is documented at the BitTorrent site here:
http://www.bittorrent.org/fast_extensions.html

Distributed Hash Table

- Reserved Bit: The last bit in the 8th reserved byte i.e. reserved[7] = 0x01

This extension is to allow for the tracking of peers downloading torrents without the use of a standard tracker. A peer implementing this protocol becomes a "tracker" and stores lists of other nodes/peers which can be used to locate new peers.

The specification is documented at the BitTorrent site here:
http://www.bittorrent.org/Draft_DHT_protocol.html

Connection Encryption

This extension allows the creation of encrypted connections between peers. This can be used to bypass ISPs throttling BitTorrent traffic.

The specification is documented at
http://www.azureuswiki.com/index.php/Message_Stream_Encryption

The documentation is fairly complete, but ideally it would be clarified on several points including guidance on when encrypted connections should be attempted, fallback procedures to regular connections etc.

Unofficial Extensions To The Protocol

WebSeeding

The possibility to seed a torrent via a web server is generally called WebSeeding. It allows the HTTP server to work as a peer in the BitTorrent network.

There are at least two specification for how to combine a torrent download with a HTTP download. The first standard, implemented by BitTornado is quite easy to implement in the client, but is intrusive on the HTTP in that it requires a script handling requests on the server side. i.e. A plain HTTP server that just serves plain files isn't enough. The benfits is that the script can be more abuse resistant. This specification is found here: http://bittornado.com/docs/webseed-spec.txt

The second specification requires slightly more from the client, but downloads from plain HTTP servers. It is specified here: http://www.getright.com/seedtorrent.html. It has been implemented by GetRight and libtorrent.

Extension protocol

- Reserved Bit: The fourth most significant bit in the 6th reserved byte i.e. reserved[5] = 0x10

This is a protocol for exchanging extension information and was derived from an early version of azureus' extension protocol. It adds one message for exchanging arbitrary handshake information including defined extension messages, mapping extensions to specific message IDs. It is documented here: http://www.rasterbar.com/products/libtorrent/extension_protocol.html and is implemented by libtorrent, uTorrent and Mainline.

Reserved Bytes

The reserved bits are numbered 1-64 in the following table for ease of identification. Bit 1 corresponds to the most significant bit of the first reserved byte. Bit 8 corresponds to the least significant bit of the first reserved byte (i.e. byte[0] = 0x01). Bit 64 is the least significant bit of the last reserved byte i.e. byte[7] = 0x01

An orange bit is a known unofficial extension, a red bit is an unknown unofficial extension.

Автор: gregsparrow на 3:45 PM

3 комментария:

Анонимный комментирует...

Спасибо!

5:15 PM

Анонимный комментирует...

Отличяная статья, мне нравится, одстойно.

2:21 AM

Анонимный комментирует...

[url=http://27kadrov.ru/index.php?newsid=588]смерть в эфире смотреть онлайн[/url]
[url=http://27kadrov.ru/index.php?newsid=511]онлайн 100 футов[/url]

С самого момента своего появления качественное кино приобрело всемирную популярность, и стало настоящим искусством. К нему пытались приобщиться все. С течением лет искусство кинематографии исключительно развивалось и процветало, создавались совсем новые жанры и стили, способы создания фильмов, невероятные спецэффекты. Можно отметить, что сегодня кинокартины находятся на пике своей популярности, и в спросе как старые кинокартины, так и новые фильмы, только что вышедшие в свет, которые поражают кинолюбителя интересной развязкой и своими спецэффектами. Но наибольшим преимуществом развития киноиндустрии является отсутствие необходимости платить немаленькие деньги за билет в кино либо тратиться на дорогой диск с кино, чтобы посмотреть в

хорошем качестве. Сейчас все немного проще, достаточно зайти на страницу 27kadrov.ru, чтобы посмотреть необходимый фильм онлайн полностью бесплатно, не платя за это удовольствие ничего, не отсылая никаких смс, не проходя регистрацию.
На 27kadrov.ru представлены картины разного жанра, которые смогут удовлетворить вкусы любого зрителя. На этом ресурсе можно посмотреть даже самые последние кинофильмы, в отличном качестве с высокой скоростью.

[url=http://www.27kadrov.ru/index.php?newsid=659]посмотреть дети шпионов 4[/url]
[url=http://www.27kadrov.ru/index.php?newsid=601]фильм скачать власть убийц[/url]

3:44 AM

Отправить комментарий

Следующее Главная страница

Подписаться на: Комментарии к сообщению (Atom)