УДК 004

А.Григоращенко, магістр гр. КІ-22М-2,
Центральноукраїнський національний технічний університет

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ МЕНЕДЖЕРА ЗАВАНТАЖЕНЬ З ВИКОРИСТАННЯМ ПРОТОКОЛУ BITTORRENT

У статті розроблено програмне забезпечення, яке призначено для системи менеджера завантажень з використанням протоколу BitTorrent. Метою розробки є дослідження та програмна реалізація системи менеджера завантажень з використанням протоколу BitTorrent. Об'єктом дослідження є процес менеджера завантажень з використанням протоколу BitTorrent. Предметом дослідження є методи менеджера завантажень з використанням протоколу BitTorrent. Методи дослідження базуються на методах теорії телекому, методах математичної статистики, методах розробки програмного забезпечення. Результат роботи – програмна реалізація системи менеджера завантажень з використанням протоколу BitTorrent. В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Постановка проблеми. Торрент (torrent) – це мережний протокол для обміну файлами. Файли розбиваються на невеликі частини й у такому виді передаються по мережі. Торрент-клієнт (torrent-client) завантажує ці частини й потім збирає в себе файл воєдино. У процесі завантаження шматочків клієнт також віддає вже заколисані частини, що дозволяє передавати торренти з великою швидкістю й без очікування звільнення джерела (сідера, seed).

Для того щоб скачати торрент файл, клієнт з'єднується з торрент трекером (torrent tracker), передає йому інформацію про свою IP адресу й хеш суму файлу, що потрібно скачати. Трекер відправляє клієнтові IP адреси інших клієнтів, що також роздають або качають торрент. У процесі завантаження клієнт регулярно спілкується із сервером, повідомляючи інформацію про завантаження й одержуючи оновлений список IP адрес.

Клієнти передають інформація прямо між собою без участі торрент-трекеру. Трекер тільки збирає дані із клієнтів про процес завантаження, підключених клієнтах і іншій інформації. Для оптимальної роботи торрент протоколу потрібно, щоб максимальна кількість клієнтів могли приймати й віддавати файли. При некоректному налаштуванні міжмережний екран/брандмауера або трансляції адрес/NAT, швидкість передачі може значно зменшитися або припинитися зовсім.

Аналіз останніх досліджень і публікацій. При аналізі останніх досліджень і публікацій [1-20] було виявлено певні прогалини у забезпеченні системи менеджера завантажень з використанням протоколу bittorrent.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи менеджера завантажень з використанням протоколу BitTorrent.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем менеджера завантажень з використанням протоколу BitTorrent.

– Дослідження системи менеджера завантажень з використанням протоколу BitTorrent.

– Програмна реалізація системи менеджера завантажень з використанням протоколу BitTorrent.

Об'єктом дослідження є процес менеджера завантажень з використанням протоколу BitTorrent.

Предметом дослідження є методи менеджера завантажень з використанням протоколу BitTorrent.

Методи дослідження базуються на методах теорії телекому, методах математичної статистики, методах розробки програмного забезпечення.

Виклад основного матеріалу. Специфікація протоколу BitTorrent v 1.0 .

Ідентифікація. BitTorrent – це протокол для поширення файлів, заснований на принципі " крапка-крапка" і розроблений Бремом Кохеном (Bram Cohen). Відвідаєте його сторінку за адресою http://www.bittorrent.com. BitTorrent розроблений для полегшення передачі файлів безлічі пірів по ненадійних мережах.

Ціль цієї специфікації полягає в тому, щоб зародзілювати у деталях специфікацію протоколу BitTorrent версії 1.0. Сторінка специфікації протоколу Брема обмежена основними поняттями, і втрачає супутні деталі в деяких областях. Я сподіваюся, що цей розділ буде написаний у ясних і однозначних визначеннях, які можуть послужити основою для обговорень і реалізації в майбутньому.

Предметна область

Цей розділ відноситься до першої версії (тобто версії 1.0) специфікації протоколу BitTorrent. На даний момент, він відбиває файлову структуру торрентів, протокол обміну даними між пірами (peer wire), і специфікації HTTP/ HTTPS-трекерів. З появою нових версій цих протоколів, вони повинні бути специфіковані на своїх окремих сторінках, але не тут.

У цьому розділі використовується ряд умовних позначок з метою представити інформацію в короткому й однозначному виді:

– Peer vs клієнт: У цьому розділі peer – це будь-який ***клієнт*** (це слово варто замінити адекватним синонімом) BitTorrent, що бере участь у завантаженні. Клієнт – це теж peer, однак це BitTorrent-клієнт, запущений на локальній машині. Читачі цієї специфікації можуть думати про нього як про клієнта, з'єднаним з деяким числом пірів.

– Шматок vs блок: У цьому розділі шматок відноситься до деякої порції завантажених даних, що описана в метафайлі, і яка може бути перевірена за допомогою хешу SHA1. Блок – це порція даних, що клієнт може запросити в піру. Два або більша кількості блоки становлять шматок, що потім може бути перевірений.

– Стандарт "де-факто": більші блоки тексту курсивом описують загальноприйняті підходи в різних реалізаціях BitTorrent-клієнтів, що стали стандартом "де-факто".

– Bencoding – це спосіб визначення й організації даних у стислому форматі. Він підтримує наступні типи: байтові рядки (byte strings), цілі числа (integers), списки (lists) і хеш-таблиці (dictionaries).

– Байтові рядки кодуються в такий спосіб: <довжина рядка, кодуєма в десяткової ASCII, у якій є тільки символи цифр>:<строкові дані>.

Треба помітити, що тут немає фіксованого початкового й кінцевого роздільника.

Приклад:

4:spam представляє рядок "spam"

Цілі числа кодуються в такий спосіб: i<ціле число, закодоване в десяткової ASCII>e

Початковий символ і і кінцевий е – початковий і кінцевий роздільники відповідно. Ви можете кодувати негативні числа, такі як i-3e. Цілому числу не може передувати префікс, що складається з нулів, як наприклад i04e. Разом з тим, i0e є є коректним записом нуля.

Приклад:

i3e представляє число "3".

Максимальна кількість біт цілого числа не специфікується, але для розгляду "більших файлів" aka .torrent для більше 4-х гігабайт необхідно знакове 64-бітне ціле.

Списки

Списки кодуються в такий спосіб:

l< bencode-закодовані значення>e

Початковий символ l і кінцевий e – початковий і кінцевий роздільники відповідно. Списки можуть містити bencode-кодовані значення будь-яких типів, включаючи цілі числа, рядки, хеш-таблиці й інші списки.

Приклад:

l4:spam4:eggse представляє список із двох рядків:

["spam", "eggs"]

Хеш-таблиці кодуються в такий спосіб:

d< bencode-кодована рядок-ключ>< bencode-кодований елемент>e

Початковий символ d і кінцевий e – початковий і кінцевий роздільники відповідно. Помітьте, що ключі повинні бути bencode-кодованими рядками. Значення хеш-таблиці можуть бути bencode-кодованими значеннями будь-якого типу, включаючи цілі числа, рядки, списки й інші хеш-таблиці. Ключі можуть бути тільки строковими й повинні фігурувати у відсортованому порядку (sorted as raw strings, not alphanumerics). Рядки повинні рівнятися з використанням бінарного порівняння, а не культурно^-специфічного "природного" порівняння.

Приклад:

d3:cow3:moo4:spam4:eggse являє собою хеш-таблицю { "cow" => "moo", "spam" => "eggs" }

Приклад:

d4:spaml1:a1:bee представляє хеш таблицю виду { "spam" => ["a", "b"] }

Приклад:

d9:publisher3:bob17: publisher-webpage15:www.example.com18:publisher.location4:homee представляє хеш-таблицю { "publisher" => "bob", " publisher-webpage" => "www.example.com", "publisher.location" => "home" }

Реалізація мовою Perl:

http://search.cpan.org/dist/ Convert-Bencode/lib/Convert/Bencode.pm

Реалізація мовою Java:

http://www.koders.com/java/fid47111A56F2466C232E09AEF75A39915EC70D3536.aspx52

Структура файлу позначка-даних

Файл позначка-даних закодований в bencode-форматі, докладний опис якого наведено вище.

Уміст файлу позначка-даних (розширення файлу – ".torrent") – це bencode-кодована хеш-таблиця, що містить перераховані нижче ключі. Всі строкові величини закодовані в UTF-8:

– info: Хеш-таблиця, що описує файл(и) торренту. Є дві можливих форми: перша – для випадку з 'одне-файловим' торрентом, без опису структури директорій; друга – для 'багато-файлового' торренту;

– announce: URL трекеру для публікації торренту (рядок);

– announce-list: (опціональний) Це розширення до офіційної специфікації зі збереженням зворотної сумісності. Ключ використовується для реалізації списку резервних трекерів. Повний опис можна знайти тут – http://home.elp.rr.com/tur/ multitracker-spec.txt;

– creation date: (опціональний) Дата створення торренту, у стандартному форматі UNIX-Часу (ціле число секунд минулих з 01 січня 1970 00:00:00 по UTC);

– comment: (опціональний) Текстовий коментар у вільній формі від автора (рядок);

– created by: (опціональний) Ім'я й версія програми, що використовувалася для створення torrent-файлу (рядок).

Цей розділ описує загальні поля й для "одне-файлового", і для "багато-файлового" торренту:

– piece length: Розмір кожного шматка в байтах (ціле);

– pieces: Рядок, складений об'єднанням 20-байтових значень SHA 1-хешів кожного шматка (один шматок – один хеш) (байтовий рядок);

– private: (опціональний) Це поле є цілим числом. Якщо воно встановлено в значення "1", клієнт ЗОБОВ'ЯЗАНИЙ одержувати список пірів ТІЛЬКИ за допомогою трекерів, перерахованих у файлі позначка-даних. Якщо поле встановлене в "0" або взагалі відсутній, клієнт може одержувати список пірів іншими способами, наприклад за допомогою PEX (обмін пірами) або DHT. Таким чином, слово "приватний" можна читати як "без зовнішніх джерел списку пірів".

Не всі згодні з таким описом. От наприклад визначення з вики клієнта

Azureus: http://www.azureuswiki.com/index.php/Secure_Torrents.

Крім того, слід зазначити, що навіть якщо це поле й використовується на практиці, воно не є частиною офіційної специфікації.

Info в одне-файловому режимі

У випадку одне-файлового режиму, хеш-таблиця info доповнюється наступними ключами:

– name: Ім'я файлу, що містить торрент. Носить рекомендаційний характер. (рядок);

– length: Розмір файлу в байтах (ціле);

– md5sum: (опціональний) 32-символьна шістнадцятковий рядок відповідна MD 5-сумі файлу. Вона не використовується в BitTorrent, але записується в торрент деякими програмами для кращої сумісності.

Info у багато-файловому режимі

У випадку багато-файлового режиму, хеш-таблиця info доповнюється наступними ключами:

– name: Ім'я кореневої директорії, що містить торрент. Носить рекомендаційний характер. (рядок);

– files: Список з хеш-таблиць, по однієї на кожний файл. Кожна хеш-таблиці в цьому списку містить наступні ключі:

– length: Розмір файлу в байтах (ціле);

– md5sum: (опціональний) 32-символьна шістнадцятковий рядок відповідна MD 5-сумі файлу. Вона не використовується в BitTorrent, але записується в торрент деякими програмами для кращої сумісності;

– path: Список, що містить один або більше строкових елементів, об'єднання яких дає шлях і ім'я файлу. Кожний елемент у списку відповідає або ім'я директорії, або (у випадку з останньому елементом) – ім'я файлу. Наприклад, файл "dir1/dir2/file.ext" повинен складатися із трьох строкових елементів: "dir1", "dir2" і "file.ext". Він кодується як список з рядків в bencode-форматі от так:

l4:dir14:dir28:file.exte

Ключ 'piece length' установлює номінальний розмір шматка, що, як правило, кратний 2-м. Розмір шматка звичайно вибирається виходячи із загальної кількості файлових даних у торренті, зважаючи на те, що занадто великий розмір шматка неефективний, а занадто малий дає в результаті великий торрент файл. При виборі найменшого розміру шматка керуйтеся здоровим глуздом. Бажано не робити торрент-файл розміром більше, ніж 50-75КБ (щоб полегшити його завантаження на сервер). Однак, зараз, коли розміри хостинга й ширина каналів не сильно обмежуються, кращий розмір шматка для більше ефективної роздачі файлів дорівнює 512КБ або менше (принаймні для торрентів приблизно до 8-10ГБ), навіть якщо це приведе до збільшення торрент-файлу. Часто використовувані розміри – 256КБ, 512КБ і 1МБ. Кожний шматок дорівнює обраному розміру, за винятком останнього, розмір якого може бути менше. Кількість шматків обчислюється розподілом загального розміру файлів торренту на розмір шматка й округляється в більшу сторону. Для обчислення границь шматків у багато-файловому торренті, файлові дані розглядаються як один великий безперервний потік, складений об'єднанням умісту всіх файлів у порядку їхнього проходження в списку 'files'. Число шматків і їхніх границь визначаються в такий же спосіб,

як і у випадку одного файлу. Шматки можуть перекривати границі файлів (тобто початок шматка може перебуває наприкінці попереднього файлу, а кінець – на початку наступні).

Кожний шматок має відповідний йому SHA 1-хеш. Для формування значення ключа 'pieces' (див. опис хеш-таблиці 'info' вище) хеши всіх шматків об’єднуються в один рядок (зверніть увагу – у рядок, а не в список). Її довжина повинна бути кратна 20-ти.

Протокол трекеру (HTTP/HTTPS)

Трекер – це HTTP/HTTPS сервіс, що відповідає на HTTP GET запити. Запити містять у себе метрику від клієнтів, що допомагає трекеру вести статистику для торренту. У відповідях утримується список пірів, щоб клієнт міг брати участь у роздачі. Базовий URL запиту складається з 'announce URL', що визначений у файлі позначка-даних (торрент-файл), і параметрів, які додаються до цьому URL за допомогою стандартного CGI-Методу (т.е. додавання '?' після announce-URL, за яким треба послідовності 'параметр=значення', розділені символом '&').

Зверніть увагу, що всі бінарні дані в URL (особливо info_hash і peer_id) повинні бути правильно екрановані. Це означає, що будь-який байт, що не входить у безлічі " 0-9", " a-z", " A-Z" і "$-_.+!*'()," повинен бути закодований у форматі "%nn", де nn – шістнадцяткове значення байта. (см. RFC 1738 для подробиць).

Для наступного 20-байтового хешу:

\x12\x34\x56\x78\x9a\xbc\xde\xf1\x23\x45\x67\x89\xab\xcd\xef\x12\x34\x56\x78\x9a

правильно закодованої є рядок

%124Vx%9A%BC%DE%F1%23Eg%89%AB%CD%EF%124Vx%9A

Параметри запиту до трекеру

Далі треба опис параметрів, використовуваних в GET запиті від клієнт до трекеру:

– info_hash: 20-байтовий SHA 1-хеш від значення ключа 'info' файлу позначка-даних, що є хеш-таблицею в bencode-форматі. Опис 'info' було дано раніше;

– peer_id: 20-байтовий рядок, що використовується як унікальний ідентифікатор клієнта, згенерований ним же при запуску. Значення може бути кожним, у тому числі й бінарним. На даний момент немає ніяких рекомендацій з генерації цього ідентифікатора. Однак, справедливо припустити, що він повинен бути унікальним для локальної машини. Таким чином, імовірно, варто включати в нього таку інформацію, як ідентифікатор процесу й, можливо, тимчасову мітку, записану їм при запуску. Способи кодування цього поля основними клієнтами опис нижче в розділі peer_id;

– port: Номер порту, що прослуховує клієнт. Стандартні порти, які зарезервовані для BitTorrent – 6881-6889. Клієнт може використовувати будь-який інший порт, якщо він не може відкрити його в зазначеному діапазоні;

– uploaded: Сумарна кількість відданих даних (після того, як клієнт послав подію 'started' трекеру) записане десятковим числом. Поки це точно не визначено в офіційній специфікації, уважається, що тут повинне бути загальне число відданих байт;

– downloaded: Сумарна кількість завантажених даних (після того, як клієнт послав подію 'started' трекеру) записане десятковим числом. Поки це точно не визначено в офіційній специфікації, уважається, що тут повинне бути загальне число завантажених байт;

– left: Число байт десятковим числом, що клієнт ще повинен скачати;

– compact: Устанавленное значення "1" сигналізує, що клиет може приймати компактні відповіді. Список пірів заміняється рядком – по 6 байт на одного піру. Перші чотири байти – це хост (у мережному порядку байтів), останні два байти – порт (знову ж, у мережному порядку байтів). Варто пам'ятати, що деякі трекери підтримують тільки компактні відповіді (для економії трафіку) і ігнорують запити без параметра "compact=1" або просто посилають компактну відповідь, навіть при "compact=0";

– no_peer_id: Говорить про те, що трекер може зневажити полем 'peer id' у хеш-таблиці 'peers'. Цей параметр ігнорується, якщо включено компактний режим;

– event: Значенням може бути 'started', 'completed', 'stopped', або порожнє, що рівнозначно невизначеному. Якщо параметр не визначений, значить цей запит виконується через регулярні інтервали часу. Докладніше про значення:

– started: Перший запит до трекеру обов'язково повинен бути з параметром "event=started";

– stopped: Повинен бути посланий трекеру, якщо клієнт правильно завершує роботу;

– completed: Повинен бути посланий трекеру при завершенні завантажування. Однак ця подія не повинне посилати, якщо при запуску клієнта завантажування вже на 100% завершена. Очевидно, це потрібно для того, щоб дати можливість трекеру правильно збільшувати показник завершених завантажувань, що залежить від цієї події;

– ip: (опціональний) Реальний IP-адреса клієнтської машини, формат адреси – чотири байти (десятковими числами) розділених крапками або шістнадцяткова IPv6-адреса (RFC 3513). Взагалі, цей параметр не є необхідним, тому що адреса клієнта може бути взятий з IP-адреси, з якого відправлений запит. Параметр потрібний тільки у випадку, коли IP-адреса, з якого прийшов запит, не є IP-адресою клієнта. Це відбувається, коли клієнт з'єднується із трекером через проксі-сервер. А також, це необхідно, коли клієнт і трекер перебувають в одній локальній частині NAT-шлюзу, т.к. інакше трекер буде видавати внутрішній (RFC 1918) адреса клієнта, що не є маршрутизуємим. Тому, клієнт повинен однозначно встановити IP-адресу (зовнішній, маршрутизуємий), для видачі його зовнішнім пірам. Різні трекери обробляють цей параметр по-різному. Деякі приймають його, якщо IP-адреса, з якого прийшов запит, перебуває в діапазоні RFC 1918, інші – приймають беззастережно, треті – повністю його ігнорують. Якщо передано IPv 6-адру (наприклад, 2001:db8:1:2::100), значить клієнт може спілкуватися тільки по протоколі IPv6;

– numwant: (опціональний) Кількість пірів, що клієнт хоче одержати від трекеру. Значення може бути нулем. Якщо параметр не заданий, по-умовчанню, звичайно віддається 50 пірів;

– key: (опціональний) Додаткова ідентифікація, що не доступна іншим користувачам. Призначена для того, щоб клієнт міг підтвердити свою дійсність при зміні IP-адреси;

– trackerid: (опціональний) Якщо попередня відповідь містила значення 'tracker id', це значення потрібно вписати сюди.

Трекер відповідає текстом (text/plain), що містить у собі хеш-таблицю в bencode-форматі з наступними ключами:

– failure reason: Якщо є присутнім, то є єдиним ключем у хеш-таблиці. Значення ключа – це текстове повідомлення про помилку, що повідомляє про те, чому запит не вдався (рядок);

– warning message: (новий, опціональний) Схожий на 'failure reason', але відповідь буде повним. Попередження відображається також, як і помилка;

– interval: Інтервал у секундах, що клієнт повинен витримувати між посилкою регулярних запитів трекеру;

– min interval: Мінімальний інтервал для оповіщень. Якщо задано, клієнт не повинен робити оповіщення частіше, ніж це значення;

– tracker id: Рядок, що клієнт повинен посилати назад у наступних оповіщеннях. Якщо попереднє оповіщення містило 'tracker id', а в поточній відповіді ключ відсутній, використовуйте старе значення;

– complete: Число пірів, що мають всі файли торренту. Їх називають сідерами (ціле)

– incomplete: Число пірів, що не має всі файли торренту. Їх називають личерами (ціле)

– peers: (модель на хеш-таблицях) Значенням є список, що складається з хеш-таблиць, кожна з яких містить наступні ключі:

– peer id: Ідентифікатор піру для запитів трекеру, що він сам собі й вибрав. Був описаний раніше (рядок);

– ip: IP-адреса піру у форматі IPv6 або IPv4, або DNS-Ім'я (рядок);

– peers: (бінарна модель) Замість використання хеш-таблиць, значенням кожного елемента списку може бути рядок, що складається з 6 байт. Перші 4 байти – це IP-адреса; останні 2 байти – порт. Всі байти записуються в мережному порядку байтів (big endian нотація).

Як згадувалося раніше, список пірів, по-умовчанню, має 50 записів. Якщо торрент має невелика кількість пірів, список буде менше. У противному випадку, трекер вибирає піри для списку випадковим образом. Для здійснення вибірки пірів для списку, трекер може використовувати більше інтелектуальний алгоритм. Наприклад, не повідомляти про наявних на роздачі сідерах іншим сідерам.

Клієнти можуть посилати запити трекеру частіше, ніж через зазначений інтервал: якщо відбулася яка-небудь подія (наприклад, зупинка (stopped) або завершення (completed) завантажування), або клієнт хоче одержати ще один список пірів. Проте, постійне опитування трекеру (в оригіналі, hammer – бити, наносити удари) для одержання списків пірів – це погано. Якщо клієнт хоче одержати список більшого розміру, йому варто використовувати в запиті параметр 'numwant'.

Примітка розроблювача: Навіть 30 пірів досить. Насправді, офіційний клієнт 3-їй версії створює нові з'єднання, тільки якщо має менш 30 пірів, і відмовляє в з'єднанні, при більш ніж 55 пірах. Це значення має велике значення для продуктивності. Коли новий шматок повністю отриманий, більшості активних пірів повинне бути послане HAVE-повідомлення. У результаті, кількість трафіку збільшується пропорційно кількості пірів. Якщо їх більше 25-ти, досить малоймовірно, що нові піри піднімуть швидкість завантаження. Розроблювачам клієнтів настійно рекомендується зробити так, щоб цей параметр був непомітний і складний для зміни, тому що він буде корисний у рідких випадках.

Метод scrape

Scrape – збирати, скребти, зскрібати.

Більшість трекерів підтримують іншу форму запиту, що використовується для одержання інформації з певного торренту (або всіх торрентів), якими управляє трекер. Замість незручного парсингу сторінки зі статистикою, клієнт може відправити такий запит, і трекер відповість так званою scrape-сторінкою.

Для запиту scrape-сторінки клієнт використовує HTTP GET метод, як у стандартного запиту описаного раннє, але по іншому URL'у. Щоб одержати scrape-url, потрібно проробити наступне. Шукаємо в announce-url останній символ '/' (слеш). Якщо текст безпосередньо наступний за '/' не 'announce', це ознака того, що трекер не підтримує scrape. У противному випадку, заміняємо 'announce' на 'scrape'.

Приклади: (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 підтримується)

~http://example.com/announce?x2%0644 -> ~http://example.com/scrape?x2%0644

~http://example.com/announce?x=2/4 -> (scrape не підтримується)

~http://example.com/x%064announce -> (scrape не підтримується)

Цей стандарт зароздільован Bram'ом у списку розсилання BitTorrent development: http://groups.yahoo.com/group/BitTorrent/message/3275

Scrape-url може бути доповнений опціональним параметром 'info_hash' з 20-байтовим значенням Це обмежить відповідь трекеру scrape-сторінкою, що буде містити інформацію тільки про запитуваний торренті. У противному випадку, статистика віддається по всім торрентам, якими управляє трекер. Якщо це можливо, для зменшення навантаження на трекер і канал, використання параметра 'info_hash' строго рекомендується.

Також, можна вказати кілька параметрів 'info_hash', якщо трекер це підтримує. Поки це не є частиною офіційної специфікації, хоча вже стало стандартом де-факто. Приклад:

http://example.com/scrape.php?info_hash=aaaaaaaaaaaaaaaaaaaa&info_hash=bbbbbbbbbbbbbbbbbbbb&info_hash=cccccccccccccccccccc

На scrape-запит трекер відповідає текстовим розділом (text/plain), іноді – стислим по методу gzip, що містить у собі хеш-таблицю в bencode-форматі з наступними ключами:

– files: Хеш-таблиця, що містить одну пару ключ/значення для кожного торренту, по якому є статистика. Якщо задано валідний параметр 'info_hash', то таблиця містить одну пару ключ/значення. Кожний ключ – це 20-байтове бінарне значення 'info_hash'. Значення ключа – це ще одна хеш-таблиця:

– complete: Число пірів, що мають всі файли торренту (сідери) (ціле);

– downloaded: Загальна кількість завершених завантажувань, зареєстрованих трекером (реєструються при події 'event=complete', тобто коли клієнт закінчила завантаження торренту);

– incomplete: Число пірів, що не має всі файли торренту (личери) (ціле);

– name: (опціональний) Внутрішнє ім'я торренту, зазначене в ключі 'name' роздягнула 'info' торрент-файлу.

Зверніть увагу, що ця відповідь має три рівні вкладених хеш-таблиць. От Приклад:

d5:filesd20:.....d8:completei5e10:downloadedi50e10:incompletei10eeee

Де ..... – це 20-байтове значення параметра 'info_hash' для торренту, зі статистикою: 5 сідеров, 10 лічерів і 50 завершених завантажувань.

Неофіційні розширення до scrape

Нижче описані ключі, які можуть використовуватися у відповіді, але їх немає в офіційній специфікації. Тому, поки вони є опціональними:

– failure reason: Текстове повідомлення про помилку, що повідомляє про те, чому запит не вдався (рядок). Клієнти, що обробляють цей ключ: Azureus;

– flags: Хеш-таблиця, що містить різноманітні прапори. Значення прапорів – це ще одна вкладена хеш-таблиця, що може містити:

– min_request_interval: Значення цього ключа – ціле число, що визначає, скільки секунд клієнт повинен чекати перед відправленням наступного scrape-запиту трекеру. Трекери, що посилають цей ключ: BNBT. Клієнти, які його обробляють: Azureus.

Протокол зв'язку між пірами (TCP)

Огляд

Протокол зв'язку між пірами (далі, просто peer-протокол) полегшує обмін шматками (pieces), перерахованих у торрент-файлі.

Зверніть увагу, що при описі peer-протоколу в оригінальній специфікації використовується термін "шматок", однак це не той "шматок", що використовується при описі торрент-файлу. Тому, у цій специфікації буде використовуватися термін "блок" для позначення даних, якими обмінюються піри по мережі.

Клієнт повинен підтримувати інформацію про стан кожного з'єднання з вилученим піром:

– choked: чи Блокує (від англ. choke – душити, віджимати) вилучений peer цього чи клієнта ні. Якщо peer блокує клієнта, це означає, що peer не буде відповідати на будь-який запит клієнта доти, поки не розблокує його. Клієнтові не слід намагатися запитувати блоки, тому що всі ці запити будуть зігноровані;

– interested: чи Зацікавлений вилучений peer у чомусь, що може запропонувати клієнт. Це означає, що вилучений peer почне запитувати блоки, коли клієнт розблокує його.

Зверніть увагу, що сам клієнт теж стежить і за тим, чи зацікавлений він у пірі (interested), а також, чи заблокований peer чи клієнтом ні (choked/unchoked). Тому, реальний список виглядає приблизно так:

am_choking: Клієнт блокує піру

am_interested: Клієнт зацікавлений у пірі

peer_choking: Peer блокує клієнта

peer_interested: Peer зацікавлений у клієнті

Клієнт починає з’єднання як "заблокований" і "не зацікавлений". Іншими словами:

am_choking = 1

am_interested = 0

peer_choking = 1

peer_interested = 0

Блок завантажується клієнтом тоді, коли він зацікавлений у пірі, а peer, у свою чергу, не блокує клієнта. Блок віддається клієнтом тоді, коли він не блокує пір, і peer зацікавлений у клієнті.

Для клієнта важливо інформувати піри про те, чи зацікавлений він у них чи ні. Інформацію про цьому варто вчасно обновляти для кожного піру, навіть якщо клієнт ним заблокований. Це дозволяє пірам знати, чи почне клієнт завантаження, коли він його розблокує (і навпаки).

Типи даних

Якщо не зазначений інший спосіб, всі цілі числа в peer-протоколі кодуються як чотирьохбайтові значення в big-endian форматі. У тому числі й префіксний розмір всіх повідомлень, які приходять після установки зв'язку.

Потік повідомлень

Peer-протокол складається з початкової установки зв'язку і наступного обміну повідомленнями із префіксним розміром

"Рукостискання" (handshake)

"Рукостискання" – це обов'язкове й перше в потоці повідомлення, що повинен передати клієнт. Його розмір – це 49 байт + довжина pstr.

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

– pstrlen: Довжина рядка <pstr> (один байт);

– pstr: Строковий ідентифікатор протоколу;

– reserved: Вісім зарезервованих байт. Всі поточні реалізації заповнюють їхніми нулями. Кожний біт у цих байтах може використовуватися для зміни режиму роботи протоколу. У своєму email Брем пропонує використовувати молодші біти, щоб старші біти можна було використовувати для зміни значення молодших;

– info_hash: 20-байтовий SHA 1-хеш ключа 'info' торрент-файлу. Це той же 'info_hash', що передається в запитах трекеру;

– peer_id: 20-байтовий рядок, використовується як унікальний ідентифікатор клієнта. Це той же 'peer_id', що передається в запитах трекеру (правда не завжди, наприклад Azareus не передає при включеній опції анонімності).

Для протоколу BitTorrent v1.0 дані такі:

pstrlen = 19 і pstr = "BitTorrent protocol".

Ініціатор з'єднання негайно посилає handshake-повідомлення. Адресат може відкласти відповідь ініціаторові, якщо він обслуговує трохи торрентів одночасно (торренти однозначно ідентифікуються по їх 'info_hash'). Незважаючи на це, адресат повинен відповісти відразу, як тільки одержить значення поля 'info_hash' у handshake-повідомленні. Трекерна функція NAT-перевірки не посилає поле 'peer_id' при рукостисканні.

Клієнт повинен обірвати з'єднання, якщо одержав handshake-повідомлення з 'info_hash' торренту, що він не обслуговує.

Якщо ініціатор з'єднання одержує handshake-повідомлення, у якому 'peer_id' не збігається з очікуваним 'peer_id', то ініціатор закриває з'єднання. Зверніть увагу, що ініціатор, очевидно, одержує інформацію про peer від трекеру, що містить у собі 'peer_id' цього піру. Тому, peer_id від трекеру й peer_id при рукостисканні повинні збігатися.

Ідентифікатор піру (peer_id)

peer_id повинен бути довжиною в 20 байт.

Є дві основних угоди кодування інформації про клієнта і його версію в peer_id: Azareus-стиль і Shadow-стиль.

В Azareus-стилі використовується наступне кодування:

'-'; два символи для ідентифікатора клієнта; чотири ASCII цифри для номера версії; '-'; випадкові числа.

Наприклад:

'-AZ2060-'...

Відомі клієнти, які використовують цей стиль кодування:

Далі йде список ID для відомих клієнтів

Клієнта, які зустрічаються в природі, але поки не ідентифіковані:

'BD' (Приклад: -BD0300-) 'NP' (Приклад: -NP0201-) 'w' (Приклад: -w2200-)

'hk' (example: -hk0010-) Chinese IP address, unrequestedly sends info dict in message 0x, reconnects immediately after being disconnected, reserved bytes = 01,01,01,01,00,00,02,01

В Shadow-стилі використовується наступне кодування: один альфа-цифровий ASCII символ, що ідентифікує клієнта; до п'яти символів для номера версії (розбивається за допомогою '-', якщо більше п'яти); три символи (звичайно ' ---', але не завжди); випадкові символи. Кожний символ у рядку з версією клієнта позначає число від 0 до 63: '0'=0, ..., '9'=9, 'A'=10, ..., 'Z'=35, 'a'=36, ..., 'z'=61, '.'=62, '-'=63.

Повний опис Shadow-стилю, включаючи інформацію про існуючі угоди, як кодувати версію трьома символами, можна знайти тут.

Наприклад:

'S58 B-B-----'... для клієнта Shadow's 5.8.11

Відомі клієнти, які використовують цей стиль кодування:

'A' – ABC

'O' – Osprey Permaseed

'Q' – BTQueue

'R' – Tribler

'S' – Shadow's client

'T' – BitTornado

'U' – UPn NAT Bit Torrent

Клієнт Bram'a зараз використовує такий стиль... 'M 3-4-2 -' або 'M 4-20-8-'.

BitComet робить щось інше. Його peer_id складається із чотирьох символів ASCII "exbc", за яким ідуть два байти X і Y, а потім випадкові символи. X – десятковий (in decimal) номер версії до коми, а Y – відповідає за два десяткових знаки після коми. BitLord використовує ту ж схему, але додає, "LORD" після байтів версії. Неофіційний патч для BitComet перемінив "exbc" на "FUTB". Кодування ідентифікаторів піру в BitComet було наведено до стилю Azureus, як в BitComet версії 0,59.

XBT Client також має свій власний стиль. Його peer_id складається із трьох заголовних символів "XBT" і потім випливають три ASCII цифри, що представляють номер версії. Якщо клієнт є відлагоджувальним складаням, сьомий байт – символ "d", у противному випадку це '-'.

Після цього треба '-', а потім випадкові цифри, заголовні й малі літери. Приклад:

"XBT054 d-d-" на початку буде означати відлагоджувальне складаня версії 0.5.4.|

Opera 8 previews і Opera 9.x releases використовують наступну схему peer_id: Перші два знаки "OP", а далі чотири цифри означають номер складаня. Всі наступні символи – випадкові шістнадцяткові цифри в нижньому регістрі.

MLdonkey використовує наступну peer_id схему: перші символи – це "-ML", далі розділений крапкою номер версії, потім "-", і далі послідовність випадкових символів. Наприклад:

'-ML2.7. 2-kgjjfkd "

Bits on Wheels використовує модель '- BOWxxx-yyyyyyyyyyyy', де y – випадковий символ (заголовна буква), а x залежить від версії. Версія 1.0.6 має XXX = A0C.

Queen Bee використовує новий стиль Брама: "Q 1-0-0 – 'або' Q 1-10-0-", а далі послідовність випадкових байт.

BitTyrant є Azureus віткою, і просто використовує "AZ2500BT '+ випадкові байти, як peer ID в 1.1 версії. Помітьте: відсутнє тире.

TorrenTopia версія 1.90 претендує бути або є похідна від Mainline 3.4.6. Його peer ID починається з '346 ---i ---i ---i ".

BitSpirit має кілька режимів peer ID. В одному режимі він читає ID пірів і переконнектується, використовуючи перші вісім байт як основу для своїх власних ID. Його реальний ID відображається з використанням '\ 0 \ 3BS "(З нотації), як перші чотири байти для версії 3.x и' \ 0 \ 2BS" – для версії 2.x. У всіх режимах ID кінці може закінчуватися як "UDP0".

Rufus використовує свою версію у вигляді десяткових ASCII значень для перших двох байт. Третій і четвертий байти – "RS". Потім випливають nickname користувача й деякі випадкові байти.

В G3 Torrent ID починається з '-G3' і додається до 9 символів nickname користувача.

FlashGet використовує Azureus стиль із "PG", але без замикаючого символу '-'. Версія 1.82.1002 – як і раніше використовує цифри версії 1.82: "0180".

BT Next Evolution походить від BitTornado, але намагається імітувати стиль Azureus. Результатом є те, що його peer ID починається з '-cB ", як і раніше з 4-значним номером версії, а потім триває трьома символами, які описують тип клієнта в стилі Shad0w peer ID.

AllPeers приймає SHA1 хеш залежний від користувача й заміняє перші кілька знаків на "ЗС" + рядок версії + "-".

Багато клієнтів використовують всі випадкові числа, або 12 нулів після випадкових чисел (наприклад, старі версії клієнта Bram'a).

Повідомлення

Всі інші повідомлення в протоколі приймають форму <length prefix><message ID><payload>. Довжина префікса складається із чотирьох байт big-endian значення. Ідентифікатор повідомлення – це один десятковий символ. Корисне навантаження (payload) безпосередньо залежить від повідомлення.

keep-alive: <len=0000>

keep-alive повідомлення – це повідомлення з нульовими байтами, length prefix установлений у нуль. Не існує ідентифікатора повідомлення й ніякого корисного навантаження повідомлення не несе. Peer може закрити з'єднання, якщо він не одержують ніяких повідомлень (keep-alive або будь-якого іншого повідомлення) протягом певного періоду часу, тому keep-alive повідомлення націлене на підтримку зв'язку. Це час, звичайно дорівнює двом хвилинам.

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

Choke-повідомлення – це повідомлення фіксованої довжини без корисного навантаження.

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

Unchoke-повідомлення – це повідомлення фіксованої довжини без корисного навантаження.

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

Interested-повідомлення – це повідомлення фіксованої довжини без корисного навантаження.

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

Non interested-повідомлення – це повідомлення фіксованої довжини без корисного навантаження.

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

Have-повідомлення фіксованої довжини. Корисне навантаження – це з вказанням нулів (zero-based) індекс шматка, що тільки що були успішно завантажений і перевірені за допомогою хешу.

Конструкторське зауваження: Це строге визначення, у реальності some games may be played. Зокрема, оскільки вкрай малоймовірно, щоб піри завантажували шматки, які вони

вже мають, peer може не рекламувати (advertise) наявність шматків пірам, які ці шматки мають. Придушення HAVE-повідомлень ("HAVE supression") як мінімум приведе до 50% скороченню числа повідомлень, а це скорочення приблизно на 25-35% накладних витрат протоколу (protocol overhead). У той же час, можливо доцільно відправити HAVE-повідомлення пірам, які вже мають цей шматок, оскільки він буде корисний у визначенні його рідкості.

Шкідливі піри також можуть вибирати оголошення (advertise) наявних шматків, які peer точно ніколи не завантажить. Due to this attempting to model peers using this information is a bad idea

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

Bitfield повідомлення може бути спрямоване відразу ж після того, послідовність "рукостискань" буде завершена, і до будь-яких інших повідомлень. Воно є необов'язковим, і клієнтам, що не має шматки, немає потреби відсилати його.

Bitfield повідомлення змінної довжини, де X – це довжина bitfield'a. Корисне навантаження повідомлення – bitfield поданя шматків, які були успішно завантажені. Старший розряд у першому байті відповідає шматку з індексом 0. Біти, які порожні вказують зниклий шматок, а встановлені біти позначають валідні й доступні шматки. Запасні біти наприкінці встановлюються в нуль.

Bitfield невірної довжини вважається помилковим. Клієнти повинні розірвати з'єднання, якщо вони одержують bitfields невірного розміру, або якщо bitfield має довільний набір запасних біт.

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

Повідомлення-Запит фіксованої довжини, використовується для запити блоку. Корисне навантаження повідомлення містить наступну інформацію:

index: ціле число, що визначає із вказівкою нулів (zero-based) індекс шматка

begin: ціле із вказівкою нулів зсув байтів усередині шматка

length: ціле число, що визначає запитувану довжину.

This section is under dispute! Please use the discussion page to resolve this!

View 1. Відповідно до офіційних специфікацій, "Всі поточних реалізацій використовують 2^15 (32KB) шматки, і закривають з'єднання, які запитують кількість даних більше 2^17 (128Kb)." Уже у версії 3 або 2004, це поводження було змінено на використання 2^14 (16Кб) блоків. Починаючи з версії 4.0 або mid-2005, з'єднання в Mainline при запитах більше, ніж 2^14 (16Кб), і деякі клієнти пішли цьому прикладу. Помнете, що block-запити менше, ніж шматки (>= 2^18 байт), тому будуть необхідні численні запити, щоб скачати весь шматок.

Властиво, специфікація дозволяє 2^15 (32Кб) запити. Реальність така, що всі клієнти починаючи із сьогоднішнього моменту будуть використовувати 2 ^ 14 (16Кб) запити. Через клієнтів, які прив'язані до такого розміру запитів, рекомендується реалізовувати програми, що роблять запити саме такого розміру. Менші розміри запитів приводять до підвищення накладних витрат у зв'язку зі збільшенням кількості необхідних запитів, проектувальники радять не робити розмір запитів менше, ніж 2 ^ 14 (16Кб).

Вибір граничного розміру запитуваного блоку не дуже ясний. Mainline версії 4 здійснює 16 Кб-Ие запити, більшість клієнтів будуть використовувати цей розмір. У той же час розмір 2^14 (16Кб) представляється підлоги-офіційним (наполовину офіційним, тому що офіційна розділяція протоколу не обновлялася), тому, по суті, неправильним (не відповідної специфікації). У той же час, дозвіл бо'льших запитів розширює набір можливих пірів, і при виключенні дуже низької пропускної здатності з'єднання (<256кб/сек), кілька блоків буде завантажене в один choke-timeperiod, у такий спосіб просте приписання старої межі розміру блоку викликає мінімальне погіршення роботи. Через цього фактора, рекомендується тільки старе 2^17 (128 КБ) максимальне обмеження розміру.

View 2. Поточна версія має принаймні наступні помилки: Mainline почали використовувати 2^14 (16384) байт запити, коли він був єдиним з існуючих клієнтів, тільки

"офіційна специфікація" усе ще говорила про застарілому 32768-байтовомц значенні, що не було в дійсності ні розміром значення за замовчуванням, ні дозволеним максимумом. У версії 4 поводження запитів не змінилося, але максимально припустимий розмір запиту став рівним значенню розміру за замовчуванням. В останній версії Mainline максимум змінився до 32768 (помітьте, що ця перша поява 32768 або для значення за замовчуванням, або для максимального розміру запиту з моменту появи першої версії). Твердження: "більшість старих клієнтів використовують 32KB запити" – є помилковим. Обговорення запитів не приймає наслідки латентності до уваги.

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

Piece-повідомлення змінної довжини, де X – довжина блоку. Корисне навантаження повідомлення містить наступну інформацію:

- index: ціле визначальне із вказівкою нулів індекс шматка;
- begin: ціле, що визначає із вказівкою нулів байтове зсув усередині шматка;
- block: блок даних, що є підмножина шматка з певним індексом.

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

Cancel-повідомлення фіксованої довжини, використовується для скасування блокування запитів. Корисне навантаження повідомлення ідентичне тої, котра була в " повідомленні-запиті" ("request" message). Повідомлення звичайно використовується під час стратегії "Кінця гри" (End game).

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

Port-повідомлення відсилається за допомогою нових версій Mainline, що реалізує DHT Tracker. Порт для прослуховування є портом який DHT вузол прослуховує. Цей peer повинен бути вставлений у локальну таблицю маршрутизації (якщо DHT Tracker підтримується).

Алгоритми

Черги

View 1. Взагалі пірам рекомендується тримати кілька невиконаних запитів для кожного з'єднання. Інакше повний круговий обіг повідомлення (туди-назад, round trip, RT) зажадає завантажити блок до завантаження нового блоку (круговий обіг PIECE-повідомлення, і далі REQUEST-повідомлення). У зв'язку з високим BDP (результат затримки смуги пропущення, високої латентності або high bandwidth), це може привести до істотної втрати ефективності.

Конструкторське зауваження: Це найбільш важливий показник продуктивності. Статична черга з 10 заявок є прийнятною для 16 Кб-их блоків при зв'язку 5 мб/сек з латентності 5 мс. Стає дуже розповсюдженим зв'язок з великою пропускною здатністю, так це підштовхує проектувальників інтерфейсів передбачати можливість змін.

View 2. більша частина інформації в розділі "Черги" є помилковою або вводить в оману. Просто до відома, що "установки за замовчуванням для 5 вихідних запитів" не можуть бути вірними в плині довгого часу, "32 KB блоки" – є помилковими, оскільки ви, як правило, не використовуєте 32 КБ блоки, і набудовуєте довжину черги, змінивши цей параметр (видимий розмір блоку) і намагаєтеся виміряти ефект, це погана ідея.

Супер-сідування

Властивість супер сід (super-seed) в і над S-5.5 є новим алгоритмом сідування, спроектованим для допомоги ініціаторові торренту з обмеженою пропускною здатністю "завантажувати" великий торрент, зменшуючи кількість обсягу даних, необхідних для вивантаження (upload) і для породження нових сідів у торренті.

Коли клієнт-сід переходить у режим "супер-сід", він не буде виступати в якості стандартного сіду, але маскується як звичайний клієнт без яких-небудь даних. Як тільки клієнти підключаться до нього, він буде інформувати їх про те, що він одержав шматок – шматок, що ніколи посилав, або, якщо всі шматки вже послав, він є дуже рідким. Це спонукає клієнта до спроби скачати тільки цей шматок.

Коли клієнт закінчив завантаження шматка, сід не буде інформувати його про будь-які інші шматки доти, поки він бачить ці шматки відіслані раніше, принаймні в одного

іншого клієнта. Доти, клієнт не буде мати доступ до будь-яких інших шматків, і тому не буде витрачатися пропускна здатність сіду.

Цей метод привів до набагато більше високої ефективності сідування, за допомогою примуса пірів в одержанні тільки найрідших даних, це й скорочення надлишкової кількості посилаємих даних, і обмеження обсягу даних, що посилаються пірам, які не сприяють поширенню цих даних (do not contribute to the swarm). До цього, сід повинен був вивантажити від 150% до 200% від загального розміру торренту перед тим як інші клієнти стають сідами. Проте, великий торрент, сідуємий єдиним клієнтом, запущеним у режимі 'супер сід' у стані зробити це тільки після 105% вивантаження даних. Це на 150-200% ефективніше, ніж при використанні стандартних сідів.

Режим 'супер-сід' не рекомендується для повсюдного використання. Хоча він допомагає в поширенні більших рідких даних, оскільки він обмежує вибір шматків, які клієнт може завантажувати, він також обмежує можливості цих клієнтів, у плані завантаження даних для вже частково отриманих шматків <з даного сіду>. Таким чином, супер-сід режим рекомендується тільки для ініціюючих сідування серверів.

Стратегія завантаження шматків

Клієнти можуть вибирати для завантаження шматки у випадковому порядку

Краща стратегія полягає в тім, щоб завантажувати найрідші шматки в першу чергу. Клієнт може визначити їх за допомогою зберігання первісного bitfield кожного піру й наступних їхніх відновлень при одержанні have-повідомлень. Потім клієнт може скачати шматки, що зустрічаються з мінімальною частотою в bitfield'ах пірів. Помітьте, що будь-яка стратегія визначення найбільш рідкого шматка (Rarest First стратегія) повинна включати елемент випадкового вибору з, принаймні, декількох найбільш рідких шматків, тому що одночасна спроба багатьох клієнтів перейти до того самого "самого рідкого" шматку є непродуктивною.

Кінець гри (End game)

Коли завантаження майже завершене, є присутньою тенденція повільного завантажування останніх декількох блоків. Для прискорення цієї операції, клієнт посилає запити про всі свої загублені блоки всім своїм пірам. Щоб це не стало жахливо неефективно, при стрибку необхідного блоку клієнт посилає cancel-повідомлення всім іншим пірам.

Існують не роздільовані граничні значення, рекомендовані відсотки, або кількість блоків, які повинні використовуватися як орієнтир або як Recommended Best Practice.

Коли переходити до стратегії "Кінець гри" – питання спірний і вимагає обговорення. Деякі клієнти переходять до стратегії "Кінець гри", коли всі шматки були запитані. Інші чекають доти, поки кількість незавантажених блоків (blocks left) стане менше, ніж кількість переданих блоків (blocks in transit, очевидно запитаних, але не прийнятих), і не більш ніж 20. Ідея зберегти кількість незакінчених блоків до 1 або 2 блоків представляється гарною для мінімізації накладних витрат, і якщо ви випадковим образом запитуєте блоки, то маєте низькі шанси скачати дублікати. Докладніше про протокольні витрати (protocol overhead), можна прочитати тут: http://hal.inria.fr/ inria-00000156/en

Блокування (Choking) і оптимістичне розблокування (Optimistic Unchoking)

Блокування (Choking, удушення, засміття) відбувається з кількох причин. Відстеження перевантажень в TCP відбувається дуже погано, коли одночасно відбувається відсилання через кілька з'єднань. Крім того блокування дозволяє кожному піру використовувати алгоритм "зуб за зуб" (for-tat-ish) для одержання доброї швидкості завантажування.

Описаний нижче алгоритм блокувань застосовується на справжній момент. Надто важливо, щоб всі наступні алгоритми однаково добре працювали й у їхніх мережах, що в повністю икористовують, і в мережах, великою частиною що використовують поточний алгоритм.

Є кілька критеріїв гарного алгоритму блокування, яким він повинен задовольняти. Він повинен обмежити число одночасних віддач для гарної продуктивності TCP. Алгоритм

повинен дозволяти уникнути частого чергування блокування й розблокування, такий механізм відомий як "фібриляція"(known as 'fibrillation'). Алгоритм припускає "оплату" пірам, які дозволяють <клієнтові> завантажувати. Нарешті, алгоритм повинен з деякою частотою випробовувати невикористовувані з'єднання, щоб перевірити краще вони використовуваних у цей час чи ні, такий механізм називається оптимістичним розблокуванням (optimistic unchoking).

Поточний алгоритм блокування уникає фибрілляції змінюючи заблоковані піри не частіше ніж раз в 10 секунд.

Піри, що мають кращу швидкість віддачі (у порівнянні зі завантажуючими), але не зацікавлені ... Коли вони стає зацікавленим, піри, що здійснюють завантаження з поганою швидкістю віддачі, відключаються. Якщо клієнт має повний екземпляр файлу, він ґрунтується на швидкості віддачі...

Офіційні розширення протоколу

На сучасний момент є кілька офіційних розширень протоколу.

Розширення "Fast Peers"

Зарезервований біт: третій біт в 8-ом захищеному байті (reserved byte), тобто reserved [7] |= 0x04

Специфікація роздільована на сайті BitTorrent тут: http://bittorrent.org/beps/bep_0006.html

Розподілені хеш-таблиці (Distributed Hash Table)

Зарезервований біт: останній біт в 8-ом захищеному байті (reserved byte), тобто reserved[7] |= 0x01

Це розширення дозволяє пірам працювати без використання стандартного трекеру. Peer, що реалізує цей протокол, сам стає "трекером" і зберігає списки інших вузлів/пірів, які можуть використовуватися для знаходження нових пірів.

Специфікація роздільована на сайті BitTorrent тут: http://bittorrent.org/beps/bep_0005.html

Шифрування з'єднання

Це розширення дозволяє створювати зашифровані з'єднання між пірами. Це може допомогти обійти обмеження, які накладають деякі провайдери інтернету на BitTorrent-трафік.

Специфікація описана в Wiki проекту Azureus: http://www.azureuswiki.com/index.php/Message_Stream_Encryption

Розділяція є досить повної, однак потрібно уточнити пари моментів щодо шифрованих з'єднань: коли вони повинні прийматися й відкіт до звичайних з'єднань у випадку невдачі при встановленні шифрованих з'єднань.

Неофіційні розширення протоколу

Протокол повідомлень Azureus

Зарезервований біт: 1.

WebSeeding

Можливість роздавати торрент через веб-сервер звичайно називається WebSeeding. Вона дозволяє HTTP-серверу виступати в ролі піру в BitTorrent мережі.

Протокол розширень

Зарезервований біт: 44, четвертий найбільш значимий біт в 6-ом зарезервованому байті, тобто reserved[5] |= 0x10.

Розширення протоколу переговорів

Зарезервовані біти: 47 і 48.

Зарезервований біт: 21.

Протокол, що враховує місце розташування піру (у географічному змісті), для кращої продуктивності. Специфікація може бути знайдена тут – http://wiki.theory.org/BitTorrent_ Location-aware_Protocol_1.0_Specification

Розробка структурної схеми

Структурна схема системи зображена на рисунку 1. З неї ми бачимо, що система представляє собою взаємодію наступних структурних блоків:

1. Інтерфейс користувача головного вікна програми.

2. Панель швидкого доступу до основних функцій менеджера завантажень з використанням протоколу BitTorrent, яке включає в себе наступні функції:

– Додати завантаження.

– Запустити завантаження.

– Зробити паузу.

– Перервати завантаження.

– Швидкість завантаження.

– Кількість одночасних завантажень.

– Посилання до форуму на сайті підтримки.

3. Вікно статусів, яке включає в себе:

– Перелік усіх завантажень.

– Категорії файлів, які завантажуються (програми, архіви, музика, відео).

– Топ завантажень (програми, архіви, музика, відео, пошук).

– Новини.

– Стан (завантаження, чекання завантаження, заплановано, помилки, пауза, завантажено).

– Історія.

– Видаленні завантаження.

4. Основна панель менеджера завантажень з використанням протоколу BitTorrent:

– Файл (топ завантажень, імпорт завантажень, імпорт, експорт, вихід).

– Завантаження (додати завантаження, додати групу завантажень, перевірити оновлення, пауза, видалити, видалити заразом з файлом, запланувати, перезавантажити заново, копіювати URL, відкрити файл, відкрити папку, скопіювати файл, перемістити файл, меню Windows, робота з архівом, коментарі, знайти дзеркала, додати в менеджер сайтів, властивості).

– Дії (стартувати все, призупинити все, тимчасова зупинка завантажуємих, видалити все, знайти, знайти далі, швидкість).

– Вид (налаштування кнопок, сортування списку, список завантажень, звук, категорії, лог завантажень, плаваюче вікно, скіни, мова інтерфейсу: українська, англійська).

– Автоматизація (стартувати усі завантаження при запуску програми, стартувати усі завантаження при появі інтернету, стартувати усі завантаження по часу, відновити зв'язок при обриві, відключитися від інтернету після завершення завантажень, перевірити завантажені файли на віруси).

– Інструменти (пошук, історія, менеджер сайтів, розклад, плагіни, налаштування: загальні, з’єднання, завантаження, проксі, автоматизація, менеджер сайтів, розклад, інтерфейс, інші, плагіни).

– Довідка (зміст, домашня сторінка, он-лайн підтримка, повідомити про помилку, форум, перевірити оновлення, про програму).

5. Вікно завантажень:

– Ім’я файлу.

– Стан завантаження.

– Розмір файлу.

– Скільки залишилося об’єму даних для завантаження файлу.

– Швидкість завантажень.

– Коментарі.

6. Блок основних функцій менеджера завантажень з використанням протоколу BitTorrent, якій розташовується у треї.

Рисунок 1 – Структурна схема системи

Висновки. У статті наведені теоретичне узагальнення й рішення наукового завдання дослідження методів менеджера завантажень з використанням протоколу BitTorrent.

Рішення даного завдання полягало у вирішенні наступних задач: Був проведений огляд існуючих систем менеджера завантажень з використанням протоколу BitTorrent; Досліджена система менеджера завантажень з використанням протоколу BitTorrent; На основі отриманих результатів досліджень створена програмна реалізація системи менеджера завантажень з використанням протоколу BitTorrent; Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання менеджера завантажень з використанням протоколу BitTorrent; Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Список літератури

1. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.
2. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.
3. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.
4. О.А. Смірнов, П.С. Усік, «дослідження перспектив використання технологічних рішень в мережах 5g» у Кібербезпека та інформаційні технології: монографія. – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.
5. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.
6. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.
7. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).
8. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139
9. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.
10. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.
11. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.
12. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.
13. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018
14. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Алгоритми формування безлічі маршрутів передачі метаданих у антивірусні хмарні системи. Збірник наукових праць "Системи обробки інформації". - Випуск 5 (142). - Х.: ХУПС - 2016. - С. 148-152.
15. Смірнов О.А., Смірнов С.А. Дідик А.К., Дрєєв О.М. Моделі системи нейромережевих експертів безпечної маршрутизації у хмарних антивірусних системах. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 36-39.
16. Смірнов О.А., Смірнов С.А., Дідик А.К., Дрєєв А.М. Спосіб контролю ліній зв'язку телекомунікаційної системи антивірусу. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. - 2016. - С. 121-127.
17. Смірнов О.А., Смірнов С.А., Дідик А.К. Метод безпечної маршрутизації метаданих у хмарні антивірусні системи. Системи озброєння та військова техніка. - Випуск 2 (46) - Х.: ХУПС - 2016. - С. 146-149.
18. Смірнов О.А., Кавун С.В., Доренський О.П., Вялкова В.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 151 с.
19. Смірнов О.А., Кавун С.В., Коваленко О.В., Дрєєв О.М. Мережні інформаційні технології. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 159 с.
20. Смірнов О.А., Кавун С.В., Коваленко О.В., Доренський О.П., Дрєєв О.М., Вялкова В.І. Комп'ютерні мережі. Навчальний посібник – Кіровоград: РВЛ КНТУ, 2016. – 233 с.
21. Смірнов О.А., Євсеєв С.П., Жукарев В.Ю., Король О.Г., Сорокін В.Є., Мелешко Є.В. Технології і стандарти комп'ютерних мереж. Навчальний посібник. – Кіровоград: КНТУ 2012. – 454 с