Статья BitTorrent ботнет - от дизайна до реализации

Пару слов о ботнетах

Все современные ботнет сети страдают одной проблемой - абузами на сервера управления, чем больше ваша сеть тем быстрее блокируют управляющий сервер. Разработчики прибегают к разным хитростям: делают списки серверов, генерацию доменов от фазы луны, fast-flux техники, .bit домены на основе криптовалют, .onion на основе тора, колхоз на p2p сетях и прочие нищебродские способы доставки управляющих серверов irc, twitter, telegram и т.п.. Большинство техник не эффективны, либо требуют носить с собой громоздкие бинари сторонних программ как в случае с тором.

Вашему вниманию господа, хочу представить новое революционное решение для нищебродов по доставке управляющих серверов вашему любимому ботнету на основе сети Bittorrent. Теперь ваши ботики никогда не потеряются навечно в гигантской сети интернет.

Всем нам известна сеть Bittorrent которую мы используем для бесплатной скачки и просмотра наших любимых сериальчиков, правообладатели этих сериалов ссут кипятком, но сделать ничего не могут, так как сеть p2p в основе которой распределённые хеш таблицы (DHT). Так почему и нам не использовать эту сеть в грязных планах, тем более она уже проверена временем и роскомнадзором.

DHT

Для начала расскажу что такое DHT и как она объединяет нас с вами при загрузке сериальчиков. Каждый пользователь сети Bittorrent имеет ряд параметров:

1. IP адрес - он может быть как внешним так и за NAT и не доступен для входящих соединений, в таком случае вы только можете качать сериальчики без возможности раздачи их.
2. Port - на который будут поступать входящие соединения.
3. ID - 20 байтный номер, который генерируется случайно один раз и сохраняется. Он будет использован в DHT для возможности связать вас с другими пользователями сети.

Итак IP, Port, ID у нас есть но как мы узнаем о других пользователях сети, ведь нам так хочется скачать сериальчик “Отбросы” (“Misfits”). Окей не проблема, умные дядьки из университета придумали базу данных с названием - распределённые хеш таблицы (DHT). Это вам не база данных как в сети bitcoin которая хранит все существующие

транзакции размером в сотню гигабайт, для скачки которой нужно ждать два дня, чтобы подключиться к сети. Эта база хранит информацию только о ближайших соседях. И речь тут идет не о твоей соседке бабе Клаве из соседнего подъезда, тут нет вообще никакого отношения к географии. Выбираются соседние ID пользователей сети. И вот простой пример:

Дано:

1. У вашего торрент клиента ID равен 17
2. В сети присутствуют такие пользователи 1000, 100, 900, 13, 1, 500, 16, 30, 19.

И мы хотим найти ближайшие три соседа к 17. Далее пойдет математика начальной школы.

1. Отсортируем числа 1, 13, 16, 17, 19, 30, 500, 900, 100, 1000. Итак кто тут три ближних соседа к 17? Догадались?
2. Результаты 1, 13, **16**, **17**, **19**, **30**, 500, 900, 100, 1000

Внимательный читатель скажет а почему 30 а не 13? А все просто у 30 дистанция поменьше

- 30-17=3
- 17-13=4

Полагаю идея понятна как выбирать соседей, но можно сделать еще удобнее и забыть о расстоянии, представьте если бы у нас ID был всегда 0 то после сортировки мы могли бы просто выбрать первые три числа и все. На помощь приходит операция XOR.

Давайте посчитаем что произойдет если все ид проксорить на 17.

17^17 = 0 - вот и стал он нулем
17^1000 = 1017
17^100 = 117
17^13 = 28
17^16 = 1
17^19 = 2
17^30 = 15
17^500 = 485
17^900 = 917

И сортируем результат

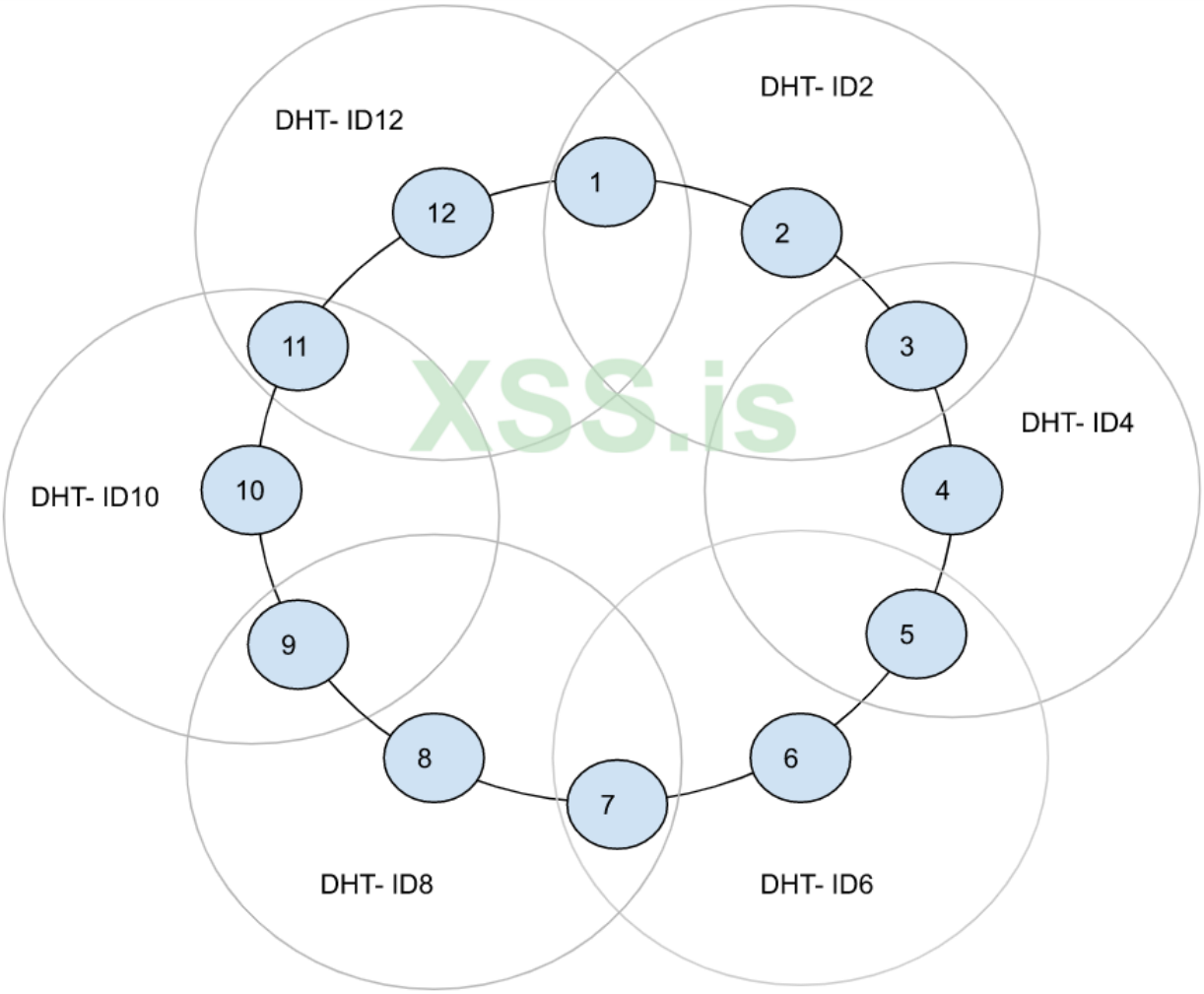
17^**17** = 0
17^**16** = 1
17^**19** = 2

17^**30** = 15
17^13 = 28
17^100 = 117
17^500 = 485
17^900 = 917
17^1000 = 1017

Заметьте результат тот же, а выбирать стало проще.

Итак если подитожить, каждый пользователь сети имеет ID, IP, Port и таблицу DHT которая хранит список своих ближайших соседей.

Диаграмма показывает как объединяются клиенты в сети



Вот так выглядит настоящая DHT у двух разных пользователей сети BitTorrent

localId: 77777777777777777777777777777777777777 ip:******* port: 6881

count: 1000
id: 777776d6ae529049f1f1bbe9ebb3a6db3c870ce1 ip: 68.41.104.165 port: 55924
id: 77777ab0d89f63726d222cb92af2c8a686842aac ip: 1.4.159.188 port: 34160
id: 777764611f7e7aaa002caae8ec0ce1d67069b9f8 ip: 190.21.148.185 port: 6881
id: 777764e5088b750652efd7e70590dcbf3d8cc507 ip: 94.60.167.189 port: 50321
id: 77775018245aaaacce53808ae8f48e746e6c4cd0 ip: 186.176.72.173 port: 25629
id: 77775f8d814e1b819965f5db178ed56c12cfba9a ip: 90.241.79.232 port: 14913
id: 77775d5ccd6e24e6e061006b43a33cf81fc6ec75 ip: 76.174.250.59 port: 30673
id: 777747e5a42325c93278b62aaeb74bd1ab5105b1 ip: 62.250.178.102 port: 6881
id: 7777424ee4ec83ae5c4cfe7af86deb558fd84dff ip: 180.158.147.191 port: 51413
id: 77774e17384763e432aa79e07a5c0e0aa81e89ae ip: 175.208.97.79 port: 11151
id: 77773125d6f0548fa2137c0072a91d548c6553dd ip: 80.68.2.19 port: 6881
id: 77773ee66a19df70e517d2203c8e15e5d4baa72f ip: 106.211.3.129 port: 48661
id: 777713d0fa1b01295f8531f9ab78f81f1883075c ip: 150.109.34.27 port: 60021
id: 77771c097ba0f68e97becdbbc420b77c0c4f6b79 ip: 176.115.144.141 port: 4227
id: 7777184a659e0903becec81d35c032722752646e ip: 176.104.207.148 port: 51413
id: 7777e3362df0c8a13fb709f7c1eb8dc7d4d13880 ip: 212.23.71.37 port: 49001
...

localId: 46ab0625e2774240a6cf65f642d6e3689bea33d5 ip: ******** port: 6881

count: 1000
id: 46abd5a4a0d9955b8d1af60850aedf9ce928eab7 ip: 181.46.147.79 port: 48075
id: 46aa6b94cd2a329194fab05c188935e4d0e73538 ip: 93.58.130.113 port: 6889
id: 46aa69a7dd0962e622ccdadc413c9444374714a1 ip: 107.171.148.47 port: 6881
id: 46aae8ffd669bcda28afe8de780ab469e9fb2752 ip: 188.242.109.172 port: 7920
id: 46a90795825730874a4f0482556a76a83e235adc ip: 185.107.71.146 port: 28007
id: 46a976300994d46d6d7c1350a91561e66dd08e71 ip: 61.80.235.33 port: 7988
id: 46a99453e74a227b56a19ffc96e34e9807a22e1f ip: 77.34.97.109 port: 6881
id: 46a9ed1ba901557c92e9a8b334478ec5a4eb1ada ip: 37.110.135.21 port: 14321
id: 46a9f81fb5ca0f7da588ac7e2da1c662127c5c20 ip: 71.231.44.150 port: 6881
id: 46a84c1578b8efcf5546191e451be903dad2a0bd ip: 179.162.250.237 port: 47278
id: 46a84cd111dc81aca8731ea0b3b9d715d85f96b9 ip: 96.33.166.33 port: 51413
id: 46a86a5a92ccd5b06462bd24dad68387cfe308f0 ip: 46.23.89.158 port: 63821
id: 46a89b467b9238e16b95d6522eaf085755be4ea0 ip: 185.107.95.77 port: 28000
id: 46af3a44788498537d498e2c731e18c3d2e61363 ip: 95.168.168.185 port: 61693
id: 46af675f13954ab184854230392d3d242aabfb44 ip: 37.200.121.86 port: 6881
id: 46ae0914774c27b5f85137c10a4a0fdb6bdd0c1a ip: 83.84.75.227 port: 50545
...

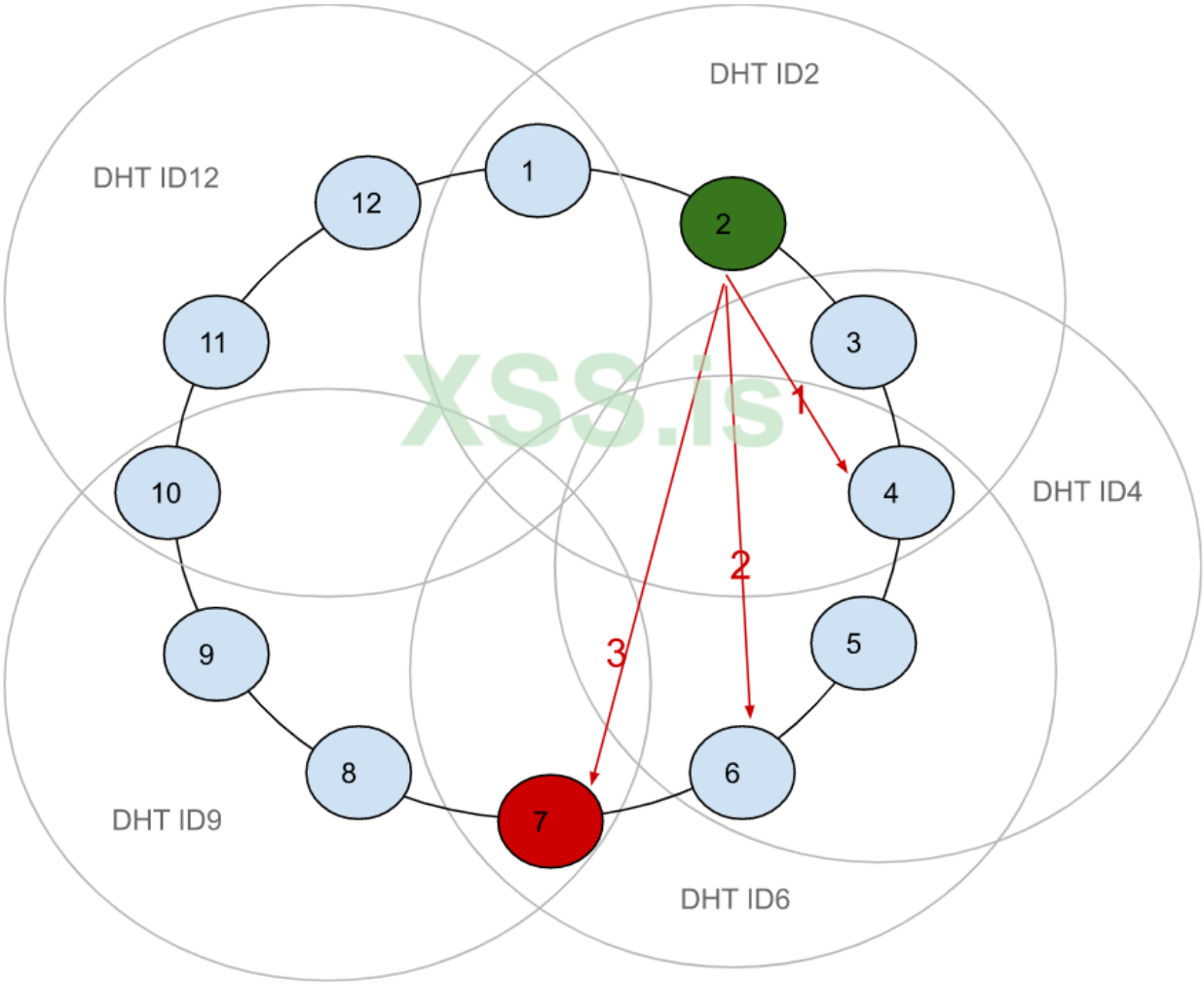
Чем больше сеть тем ID в таблицах более упорядочены.

Поиск DHT

База данных DHT не только объединяет клиентов сети, что как бы уже не плохо, но по ней еще может выполняться поиск клиента по ID. Клиент в свою очередь может хранить различную информацию, например IP и Port которые раздают ваш любимый сериальчик или хранит какую то специфичную информацию, вот ее то мы и будем использовать в наших с вами корыстных целях.

Допустим клиент ID - 2 хочет найти клиента с ID - 7, напрямую мы не можем отправить запрос клиенту 7, так как не знаем его IP и Port. Далее смотрим на диаграмму.

1. ID 2 выбирает из своей DHT таблицы [1, 2, 3, 4] ближайшего к 7, в данном случае это 4
2. ID 2 отправляет запрос в котором просит ID 4 вернуть из своей таблицы DHT список клиентов с ID близких к 7.
3. ID 4 возвращает ID2 список клиентов [6, 5, 4]
4. ID 2 выбирает из списка [6, 5, 4] близкий к 7, в данном случае это ID 6
5. ID 2 отправляет запрос в котором просит ID 6 вернуть из своей таблицы DHT список клиентов с ID близких к 7.
6. ID 6 возвращает [7, 6, 5]
7. ID 2 успешно находит IP и Port клиента с ID 7 и теперь может запросить у него дополнительную информацию.



С DHT кажется разобрались, а кто ничего не понял может удалиться с форума так как дальше пойдет просто жуть, где я подробно опишу Bittorrent протокол.

Bittorrent DHT

Список спецификаций Bittorrent можно найти на официальном сайте https://www.bittorrent.org/beps/bep_0000.html

Как вы можете заметить читать эти спеки можно до пенсии, но я тебе помогу мой начинающий ботодел, нужно прочитать только две, и мне пофиг что ты не знаешь английский:

- DHT Protocol
- Storing arbitrary data in the DHT

Bencode

Все сообщения сети сериализуются в формате Bencode. Bencode это формат данных на подобии json только поддерживает еще и бинарные данные.

Пример как будет выглядеть сообщение ошибки в сети:

```
{
    "t": "aa",
    "y": "e",
    "e": [
        201,
        "A Generic Error Ocurred"
    ]
}
```



```
bencoded = d1:eli201e23:A Generic Error Ocurrede1:t2:aa1:y1:ee
```

Для объяснения протокола Bittorrent DHT я буду в дальнейшем описывать все в формате json.

Прежде чем начать описание протокола, нужно дать описание двум форматам хранения данных, которые будут использоваться позже.

Compact nodes and peers

Список клиентов “нод” (“nodes”) хранится в специальном бинарном формате по 8и штук

Список пиров (“peers”) - те кто раздает данные, хранится в таком формате

```
//big-endian
struct Node
{
        unsigned char id[20];
        unsigned int ip;
        unsigned short port;
};
```


DHT Protocol

Все сообщения сети передаются по UDP, как было замечено выше сообщения сериализуются в bencode формате.

У каждого сообщения есть в запросе и ответе ключ “t” - transaction ID, должен быть длинной не менее 2х байт. В запросе он выбирается случайным образом, в ответ должен копироваться из запроса, это сделано для того чтобы не мешались люди - кони.

```
//big-endian
struct Peer
{
        unsigned int ip;
        unsigned short port;
};
```

Ключ “y” тоже присутствует везде и имеет такие значения:

- “q” - запрос (“query”), всегда присутствует в запросе
- “r” - ответ (“response”), всегда в ответе
- “e” - ошибка, всегда в ответе

Протокол имеет аж целых четыре команды:

- ping - проверяет жив ли клиент, или покоится с миром
- find_node - поиск клиента по ID, возвращает список 16 клиентов близких к искомому (target) из своей DHT
- get_peers - аналогичный поиску find_node, только возвращает дополнительную информацию, которая связана с ID (info_hash), а именно список IP, Port которые раздают ваш любимый сериальчик.
- announce_peer - публикует ваш IP, Port для info_hash (и роскомнадзора), сообщая тем самым что вы можете раздавать сериал.

Команда ping

Пожалуй наиболее частая в сети, проверяет жива ли нода.

Запрос:

```
{
        "t": "aa",
        "y": "q",
        "q": "ping",
        "a": {
                "id": "abcdefghij0123456789"
        }
}
```

Ключ "q" = "ping" и один аргумент “a”:[“id”:<20 байт ID>] где в качестве id - ид ноды отправителя.

Ответ:

```
{
        "t": "aa",
        "y": "r",
        "r": {
                "id": "mnopqrstuvwxyz123456"
        }
}
```

Команда find_node

Поиск target ноды по ID, возвращает список из 8 ближайших нод к target ID из своей DHT, используется для построения DHT таблицы.

Запрос:

```
{
        "t": "aa",
        "y": "q",
        "q": "find_node",
        "a": {
                "id": "abcdefghij0123456789",
                "target": "mnopqrstuvwxyz123456"
        }
}
```

Ключ "q" = "find_node" и два аргумента “a”:{“id”:<20 байт ID>,”target”:<20 байт ID>} где в качестве id - ид ноды отправителя, target - ID искомой ноды

Ответ:

```
{
        "t": "aa",
        "y": "r",
        "r": {
                "id": "0123456789abcdefghij",
                "nodes": "def456..."
        }
}
```

где id - ид ноды отвечающей, nodes - 8мь нод в формате compact nodes

Команда get_peers

Ассоциируется с torrent хешем “info_hash”, если запрашиваемая нода имеет информацию о пирах, то возвращает их IP адреса, если нет то возвращает список nodes близких к info_hash (работает аналогично find_node)

Запрос:


```
{
        "t": "aa",
        "y": "q",
        "q": "get_peers",
        "a": {
                "id": "abcdefghij0123456789",
                "info_hash": "mnopqrstuvwxyz123456"
        }
}
```

Ключ "q" = "get_peers" и два аргумента “a”:{“id”:<20 байт ID>,”info_hash”:<20 байт ID>} где в качестве id - ид ноды отправителя, info_hash - hash искомого torrent

Ответ в случае наличия на ноде пиров:


```
{
        "t": "aa",
        "y": "r",
        "r": {
                "id": "abcdefghij0123456789",
                "token": "aoeusnth",
                "values": [
                        "axje.u",
                        "idhtnm"
                ]
        }
}
```

где id - ид ноды отвечающей, values - список IP, Port нод которые раздают торрент с info_hash в compact peers формате, token - случайно сгенерированный ид, который используется в announce_peer команде.

Ответ в случае отсутствия на ноде пиров:

```
{
        "t": "aa",
        "y": "r",
        "r": {
                "id": "abcdefghij0123456789",
                "token": "aoeusnth",
                "nodes": "def456..."
        }
}
```

где id - ид ноды отвечающей, nodes - 8мь нод в формате compact nodes, token - случайно сгенерированный ид, который используется в announce_peer команде.

Команда announce_peer

Публикует пиров в сети которые будут раздавать info_hash (торрент)

Запрос:

```
{
        "t": "aa",
        "y": "q",
        "q": "announce_peer",
        "a": {
                "id": "abcdefghij0123456789",
                "implied_port": 1,
                "info_hash": "mnopqrstuvwxyz123456",
                "port": 6881,
                "token": "aoeusnth"
        }
}
```

Ключ "q" = "announce_peer", id - ид ноды отправителя, info_hash - хеш для которого хотим опубликовать peer, token - берется из get_peers, port - порт на котором наша нода будет раздавать torrent, implied_port - необязательный параметр, но если он равен 1, то значение port не учитывается, а публикуется тот порт который видит другая сторона при общении, сделано с целью обхода NAT.

Ответ:

```
{
        "t": "aa",
        "y": "r",
        "r": {
                "id": "mnopqrstuvwxyz123456"
        }
}
```


где id - ид ноды отвечающей

Bootstrap

Ну вот вроде бы четыре команды DHT protocol описал, а все еще не понятно как же выполнить первое подключение к сети? Тут все просто, в сети существуют сервера которые доступны по доменному имени:

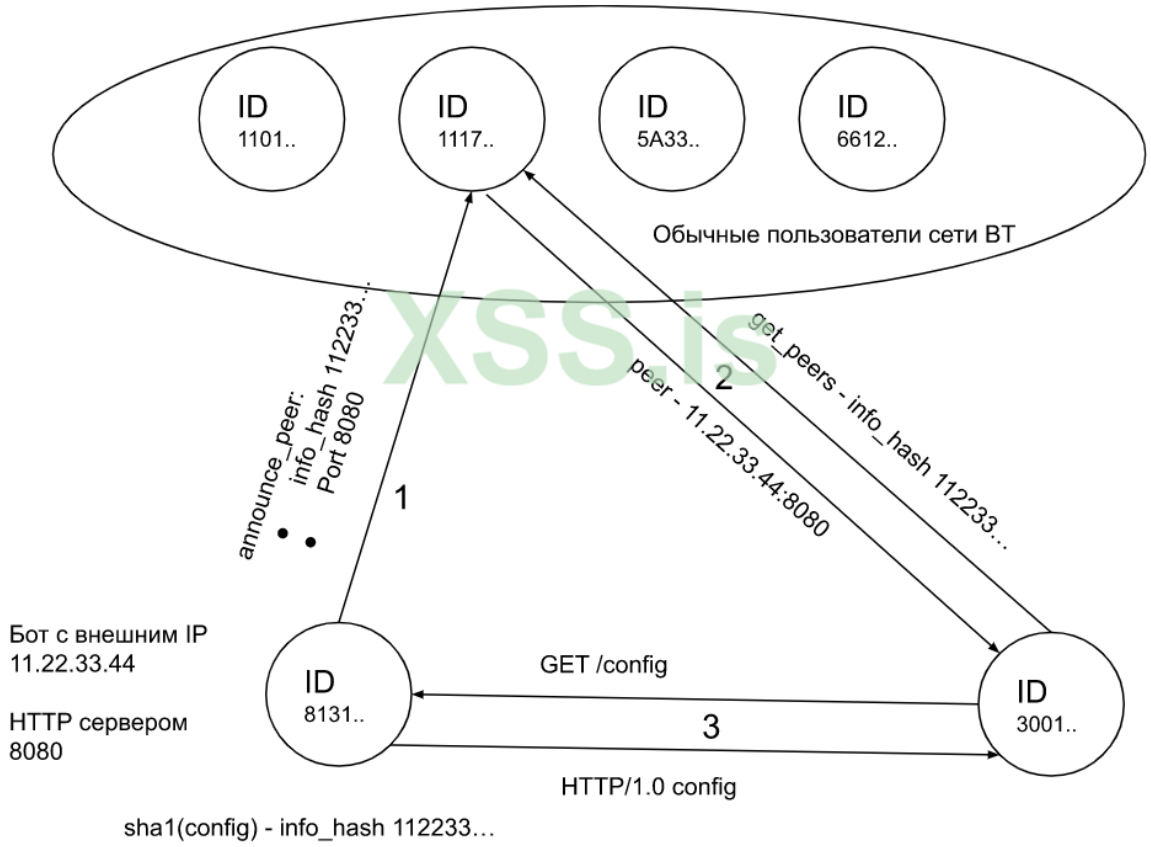
- "router.utorrent.com":6881
- "router.bittorrent.com":6881
- "dht.transmissionbt.com":6881

Они ничем не отличаются от обычных node в сети. Но для надежности лучше добавлять к этим серверам списки “хороших” нод из сети. Так как рано или поздно такие сервера могут быть заблокированы.

Стоить еще упомянуть что такое “хорошие” ноды, это те ноды с внешним IP, которые успешно отвечают на все ваши запросы в течении 15и минут.

Дизайним ботнет

Итак вроде бы все описал, а как это использовать для нашего мега-ботнета непонятно. Поможет с этим разобраться такая диаграмма.



Имеется бот ID 8131 с внешним IP 11.22.33.44, все боты имеют встроенный HTTP сервер для раздачи файлов и конфигурации на порту 8080. Бот имеет файл config - который бы хотелось раздать на другие зараженные машины.

1. Бот ID 8131.. Считает sha1 от “config”, пусть будет хеш равен 112233…
2. ID 8131.. публикует информацию о хеше 112233… в сети bittorrent на клиенте ID1117.. (имеет близкое расстояние к хешу 112233…) и свой порт 8080
3. ID1117 сохраняет info_hash 112233… и пира 11.22.33.44:8080 у себя, и будет хранить в течении двух часов.
4. Другой бот ID3001… хочет скачать файл конфигурации, он знает хеш файла.
5. ID3001… делает серию запросов get_peers к сети BT и находит ID1117.. Который возвращает peer 11.22.33.44:8080
6. ID3001… выполняет запрос к 11.22.33.44:8080 и скачивает config файл

По такой схеме коммуникации мы можем использовать публичную сеть Bittorrent для хранения информации о наших файлах, мы сохраняем info_hash файла, IP и Port ботов которые его раздают. Но в данной схеме есть один недостаток, клиент который хочет скачать файл, должен знать его хеш, а ведь мы планируем развивать свое детище и добавлять много ненужных плюшек. Что в свою очередь будет изменять info_hash. Вы скажете а почему бы нам не считать хеш не от файла, а к примеру от его имени

sha1(config), при такой конфигурации изменения контента не приведут к изменению info_hash. Но это не очень хорошая идея. Тут возникают коллизии когда в сети уже есть старая версия и мы накатываем новую, некоторые пиры будут отдавать старую версию файла и будут продолжать ее публиковать наравне с новой. Что будет тормозить процесс обновления, если не сведет его совсем на нет.

Но к счастью для решения этой проблемы есть в сети BitTorrent расширение протокола DHT Storing arbitrary data in the DHT, оно помогает публиковать любые данные в публичной сети Bittorrent и использует для этого асимметричную криптографию. Вы можете подписывать данные, и никто кроме вас не сможет их изменить.

Storing arbitrary data in the DHT

Расширение призвано сохранять произвольные данные в публичной сети DHT, вы можете сохранить данные длинной до 1000 байт, чего вполне хватает для сохранения списка IP ваших серверов управления или info_hash обновленного файла. Публикуемые данные защищаются с помощью эллиптической криптографии ed25519. Я не буду рассматривать вариант хранения неизменяемых данных (Immutable), только опишу вариант протокола для хранения изменяемых данных, так как он лучше всего подходит в наших целях.

Для поддержки сохранения произвольных данных в сети добавляется две дополнительные команды: put и get, они полностью напоминают принцип работы get_peers и announce_peer за исключением того что возвращают информацию не о пирах, а о ваших сохраненных данных.

Команда put

Запрос:

```
{
  "a":
  {
    "id": <20 byte id of sending node (string)>,
    "k": <ed25519 public key (32 bytes string)>,
    "salt": <optional salt to be appended to "k" when hashing (string)>
    "seq": <monotonically increasing sequence number (integer)>,
    "sig": <ed25519 signature (64 bytes string)>,
    "token": <write-token (string)>,
    "v": <any bencoded type, whose encoded size < 1000>
  },
  "t": <transaction-id (string)>,
  "y": "q",
  "q": "put"
}
```

Где ключ “q”=”put”, id - ноды отправителя, k - 32 байтный публичный ключ ed25519, salt - соль произвольная строка (мы будем использовать ее для именования файлов которые хотим найти), seq - монотонно увеличивающееся число, может играть версию файла в нашем случае, sig - ed25519 сигнатура, token - токен берется из get команды, v - произвольные данные которые вы хотите опубликовать в сети до 1000 байт длинной.

Ответ:

Гду id - отвечающей ноды.

```
{
  "r": { "id": <20 byte id of sending node (string)> },
  "t": <transaction-id (string)>,
  "y": "r",
}
```

Команда get

Запрос:

```
{
  "a":
  {
    "id": <20 byte id of sending node (string)>,
    "seq": <optional sequence number (integer)>,
    "target:" <20 byte SHA-1 hash of public key and salt (string)>
  },
  "t": <transaction-id (string)>,
  "y": "q",
  "q": "get"
}
```

Где ключ “q”=”get”, id - отправителя, seq - опциональный параметр, назовем его версией файла, если оно присутствует в запросе, то нода которая хранит данные, будет отдавать только данные старше значения seq, target - хеш которые мы будем искать, считается sha1(k+salt)

Ответ:

```
{
  "r":
  {
    "id": <20 byte id of sending node (string)>,
    "k": <ed25519 public key (32 bytes string)>,
    "nodes": <IPv4 nodes close to 'target'>,
    "nodes6": <IPv6 nodes close to 'target'>,
    "seq": <monotonically increasing sequence number (integer)>,
    "sig": <ed25519 signature (64 bytes string)>,
    "token": <write-token (string)>,
    "v": <any bencoded type, whose encoded size <= 1000>
  },
  "t": <transaction-id (string)>,
  "y": "r",
}
```

Где id - отправителя, k - 32 битный публичный ключ ed25519, nodes, nodes6 - список ближайших нод к target, seq - номер версии, sig - цифровая подпись ed25519 данных, v - данные размером до 1000 байт

Цифровая подпись

Для создания или подсчета цифровой подписи данные должны быть предварительно отформатированы в следующем формате

- Сериализованы в формате bencode
- Закодированные данные должны идти строго в таком порядке salt, seq, v

Пример подписи

seq:
1

value:
Hello World!

salt:
foobar

buffer being signed:
4:salt6:foobar3:seqi1e1:v12:Hello World!

public key:
77ff84905a91936367c01360803104f92432fcd904a43511876df5cdf3e7e548

private key:
e06d3183d14159228433ed599221b80bd0a5ce8352e4bdf0262f76786ef1c74d
b7e7a9fea2c0eb269d61e3b38e450a22e754941ac78479d6c54e1faf6037881d

target ID:
411eba73b6f087ca51a3795d9c8c938d365e32c1 = sha1(public key+salt)

signature:
6834284b6b24c3204eb2fea824d82f88883a3d95e8b4a21b8c0ded553d17d17d
df9a8a7104b1258f30bed3787e6cb896fca78c58f8e03b5f18f14951a87d9a08

Финальный дизайн ботнета

Итак что мы можем сделать с новыми описанными командами? Мы можем решить вышеописанную проблему с доставкой хешей обновленных файлах клиентам.

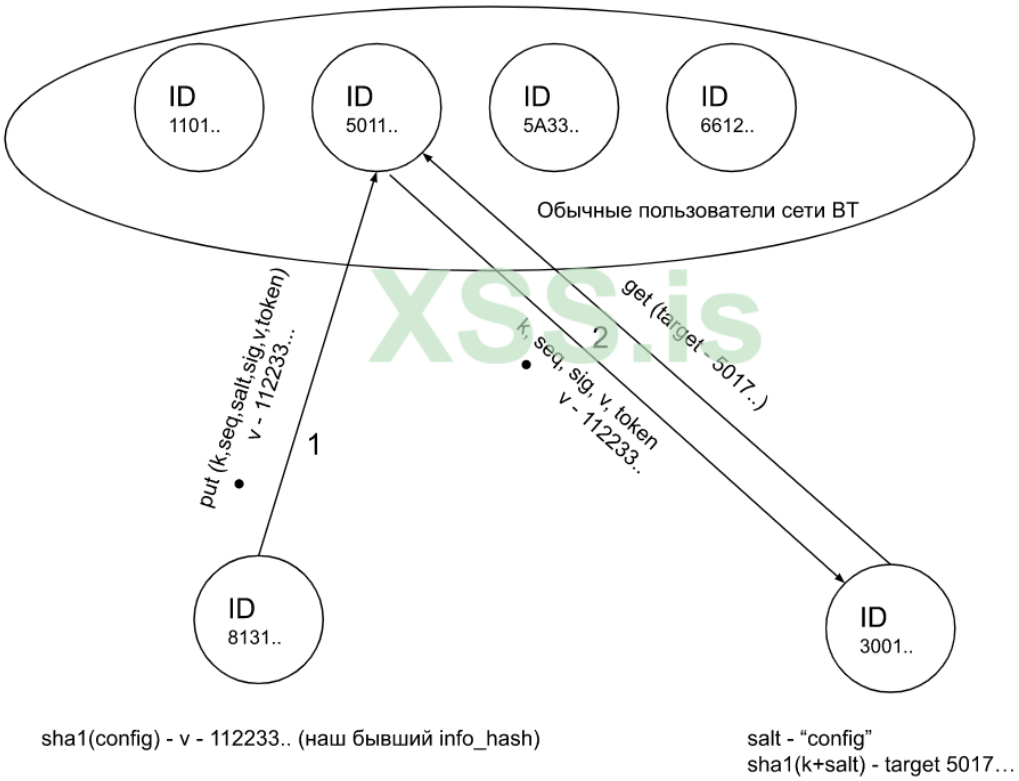
Мы будем использовать следующие параметры
Создаем пару ed25519 sk, k - секретный ключ, публичный ключ

salt - имя файла, например “config”
seq - версия файла
v - хеш обновленного файла “config”, его то мы и будем получать командой “get”
sig - ed25519_sign(bencode(salt,seq,v)) - создаем сигнатуру для нашей версии файла

Публикуем данные с помощью команды “put”

target - sha1(k+salt) считаем хеш, выполняем поиск наших опубликованных данных

Разобраться с работой нам поможет диаграмма



После получения “v”, что является info_hash мы выполняем последовательность команд из первой диаграммы, тем самым скачивая config файл.

Заключение

Разработан дизайн p2p ботнета, который использует публичную сеть DHT Bittorrent

- Ботнет не требует наличие управляющего сервера
- Все файлы обновлений и конфигураций могут быть загружены напрямую на бота с внешним IP и далее распространяться по сети.

Подход с сохранением произвольных данных в сеть BitTorrent

- Не требует наличия ботов с внешним IP
- Есть возможность публиковать произвольные данные размером до 1000 байт, это может быть список IP или доменов. Что является отличным способом доставки серверов управления вашему ботнету.

Всем спасибо и стройте неунылые ботнеты