Efficient Chunk Availability Compression for PPSP
draft-deng-ppsp-bfbitmap-01.txt

Abstract

This draft proposes to employ bloom filters in compressing chunk availability information, which is periodically exchanged between peers and the tracker through both the PPSP-TP protocol and PPSPPP protocol, so as to reduce relevant cost (in transmission, storage and computation) and enhance the overall system’s scalability.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 15, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
1. Introduction

As it is pointed out by [I-D.ietf-ppsp-problem-statement], current P2P streaming practices often use a "bitmap" message in order to exchange chunk availability. The message is of kilobytes in size and exchanged frequently, e.g., an interval of several seconds or less.

To begin with, in a mobile environment with scarce bandwidth, the message size may need to be shortened or it may require more efficient methods for expressing and distributing chunk availability information, which is different from wire-line P2P streaming.

Even in a wire-line P2P streaming application, frequent exchange of large volume of bitmap information, is among the key factors that set a limit to the system’s efficiency and scalability [P2P-limit].

Therefore, the following requirements for PPSP protocols in terms of chunk availability exchange are stated in [I-D.ietf-ppsp-problem-statement]:

PPSP.TP.REQ-3: The tracker protocol MUST take the frequency of messages and efficient use of bandwidth into consideration, when communicating chunk availability information.
PPSP.PP.REQ-7: The peer protocol MUST take the frequency of messages and efficient use of bandwidth into consideration, when communicating chunk information.

To this end, we propose to employ bloom filter algorithms in compressing chunk availability information, which is exchanged
frequently between peers and the tracker through the PPSP-TP protocol and PPSPP protocol. Given the Bloom Filters’ wide adoption in Internet and demonstrated efficiency with highly compacted data structure and low complexity and cost in terms of information storage, transportation and computation, it is expected to relieve a PPSP implementer from the dilemma between "the frequency of messages" (i.e. the timely exchange of information that contributes to better user experience) and "efficient use of bandwidth" (i.e. the limit of a single node/peer that holds the system’s overall scalability by throat).

2. Background on Bloom Filter

Bloom Filter (or BF for short) was first introduced in 1970s [BF-bloom], which makes use of multiple hashing functions to build a mapping from a set of elements to a compact binary array, to realize highly efficient member queries with a tolerably low error rate of wrongly reported hits. Despite their extraordinary efficiency in terms of storage reduction and query acceleration, BFs suffer from the fact that there is possibility that a non-member of the set be wrongly taken as a member after the query. However, research [BF-analysis] shows that the odds that a BF-based membership query makes an erroneous hit can be suppressed to near zero, by a tactful configuration of various system parameters, including the hash functions used, the number of hash functions to be used, and the length of the bit array.

```
BF(set S, integer m, hash set H)
1 filter=allocate m bits initialized to 0;
2 for each element xi in S do
3   for each hash functions hi in H do
4     filter[hi(xi)]=1;
5 return filter;
```

```
MT(element elm, BF filter, integer m, hash set H)
1 for each hash functions hi in H do
2   if (filter[hi(elm)]!=1)
3     return false;
4 return true;
```

Figure 1: Basic algorithms for BF-bitmap
As shown in Figure 1, the BF(S,m) algorithm takes a n-membered sub-set $S=\{x_1, x_2, \ldots, x_n\}$ from a universal set $U$ as input, and outputs a $m$-bit binary array $B$ as a compacted representation of $S$. In order to do that, it makes use of $k$ independent random hash functions, each of which maps a member to a marked bit in $B$ (i.e. $h_j: U \rightarrow [1,m]$, $j=1 \ldots k$). The BF algorithm is highly efficient in the following aspects:

- It is quite simple and straightforward to generate the BF representation of a set $S$, $B=BF(S)$: initially, all the bits in $B$ is set to 0; then, for each member $x$ of the set $S$, mark each bit in $B$, to which a hash function maps $x$ (as shown in Figure 1 as the BF algorithm).

- It is highly efficient to check whether or not a given element $x$ is in any BF-represented set $B=BF(S)$: for each hash function $h_j$, check the value of $B[h_j(x)]$ against 1. It is always safe to exclude the element $x$ out of set $S$, once there is a zero-valued hash bit, otherwise it is assumed that $x$ is a member of $S$ (the MT algorithm in Figure 1).

For instance, given a 2GB movie file, the original bitmap for a sharing peer would be 1024-bit (if the system is using 2MB-sized segments). By simply using 4 uniform random hash functions and a 128-bit BF-bitmap, the possibility of erroneous hits by MT algorithm would be lower than 3%.

As for a simple illustration, the 4 hash functions may be established through the MD5 message-digest algorithm [RFC1321], a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value from an arbitrary binary input. MD5 has been utilized in a wide variety of security applications, and is also commonly used to check data integrity.

Specifically, the processing of 4 hash functions is as follows: use the MD5 algorithm to turn a given chunk_ID into a 128-bit binary array, further separate the 128 bits into 4 arrays (32-bit each), and finally divide each of them using 128 to yield 4 integers in the range of $[1,m]$.

3. BF-based Chunk Availability Exchange

We first construct a general message flow (shown in Figure 2) from PPSP protocols, and then discuss how to integrate BF-bitmap algorithm with it.

3.1. A non-BF PPSP session
When a peer wants to receive streaming of a selected content (Leech mode):

1. Peer connects to a connection tracker (which may be located through a web portal) and joins a swarm.
2. Peer acquires a list of other peers in the swarm from the connection tracker (via the tracker protocol) through the CONNECT message.
3. Peer exchanges its content availability with the peers on the obtained peer list (via peer protocol) through the HAVE message.
4. Peer requests content from the identified peers (via peer protocol) through the REQUEST-DATA messages.
5. Peer periodically reports its status and chunk availability with the tracker (via the tracker protocol) through the STAT_REPORT message.
6. Peer acquires a list of other peers for a specific subset of media chunks in the swarm from the connection tracer (via the tracker protocol) through the FIND message.

3.2. A PPSP Session with BF-bitmaps
This document proposes to employ bloom filter algorithms in compressing chunk availability information exchanged and stored between peers and the tracker through the PPSP-TP protocols and PPSPP protocol. Relevant extensions to the current protocols are summarized as follows: (as shown in Figure 3)

```
+--------+      +--------+     +--------+    +--------+  +-------+
| Player |      | Peer 1 |     | Portal |    | Tracker|  | Peer 2|
+--------+      +--------+     +--------+    +--------+  +-------+

(a1)  --Page request---------------->  
      <----Page with links(+BF conf)--
      --Select stream (MPD Request)-->  
      <----------OK+MPD(x)+BF conf--

(a2)  --Start/Resume->|--CONNECT(join x)---------->

(a3)  <----------OK<--(+BF conf)+Peerlist(BF)-->  
      

(c1)  |                |<------------ HAVE(BF(S2))----------------|
      | -Get(Chunk s1)>  

(c2)  |<-------------- REQUEST(BF(s1))------------>|
      |<-----Chunk s1--|-------------------------DATA(Chunk s1)--|

(b1)  |                |-STAT_REPORT(BF(ContentMap))-->  
      |<-------------------------Ok--|

(b2)  |                |--FIND(Chunk subset S')------>

(b3)  |<----------OK+PeerList(BF)------|

Figure 3: A typical PPSP session with BF-bitmaps.
```

a. Integration to the base TP protocol
[I-D.ietf-ppsp-base-tracker-protocol]:

* (a1–a2) Configuration Setup: m, The length of the output bit array and H, the hash functions in use, are system level parameters that should be configured globally. There are two ways in achieving this: (a1) the BF configurations (or BF conf for short) be stored at the web portal and published to a requesting peer through the web page or MPD file transaction; or (a2) the BF conf be stored at the tracker and published to a joining peer through the PPSP TP protocols.
* (a3) In response to a JOIN request from a peer, the tracker may accompany the returned peer list with each recommended peer’s BF-formed chunk availability bitmap, as the initial guidance.
for the requestor to start looking for neighbors in the same
swarm. The additional cost for bearing the chunk-level
availability information is constant (O(m)) for each peer in
the returned peer list.

b. Integration to the extended TP protocol
[I-D.ietf-huang-extended-tracker-protocol]:

* (b1)STAT_REPORT: Peers use the BF(S,m,H) algorithm for
compressing the subset of locally stored and integrity
verified chunks (set S) in terms of a given swarm-ID, whenever
reporting or updating its chunk availability information with
the tracker. As the length of each BF-bitmap is constant
(O(m)), this will greatly reduce the tracker’s resource
expenditure in communicating and storing such information for a
large peer population.

* (b2-b3)FIND: Peers use the BF(S,m,H) algorithm for indicating
its query intention in the FIND request for a specific chunk
subset S’ of the given swarm to the tracker or the other
peer. The additional cost for bearing the chunk set is
constant (O(m)). In response to a FIND request with specific
chunk subset S’ in need from a peer, the tracker performs the
subset containment check on the query set parameter BF(S’)
against each registered peer’s chunk availability BF(S) by
three simple binary operations to decided whether or not to
include the peer into the peer list in return: check if "F(S)
equals (BF(S) bitwise OR (BF(S) bitwise XOR BF(S’)))" holds.
The computational cost for each subset check is constant
(O(m)).

c. Integration to the peer protocol
[I-D.ietf-ietf-ppsp-peer-protocol]:

* (c1)HAVE: Peers use the BF(S,m,H) algorithm for compressing
the subset of locally stored and integrity verified chunks
(e.g. set S2 for Peer 2 in Figure 3) in terms of a given
swarm-ID, whenever sharing its chunk availability information
with another peer. The length of each BF-bitmap is constant
(O(m)).

* (c2)REQUEST: For a downloading peer to decide which neighbor
to request for a given chunk_ID s, it uses the member query
algorithm MT(s,bf,m,H) on each neighbor’s BF-bitmap bf. The
computation cost for this member check is constant (O(m)). It
is also optional for a requesting peer to use BF-bitmap to
indicate its data request to another peer, if needed.

4. Open Issues

4.1. Algorithm Configuration Setup
As stated earlier, the BF scheme is based on a mutual arrangement between the information requestor and the responder of the basic settings for the hash algorithms (both the number of them and the specific ones) in use and the coded bitmap’s binary length. In other words, there MUST be a way of configuration setup mechanism in a local system.

To serve as the input for further discussion, we provide two initial proposals here:

- **Option1: Centralized Server for Uniform Configuration**: The most simple and straightforward way would be to set up a logically centralized configuration server to do the trick. For instance, the RELOAD base protocol introduces such a configuration server to synchronize the hash function for the P2P DHT before a peer/client joins in the overlay [I-D.ietf-p2psip-base]. There are two potential ways to integrate into the base TP protocol’s enrollment and bootstrap process: The Publishing and Searching Portal could serve as a configuration server and return the BF configuration information to the peer through player, either
  
  * via the page returned in response to a web page request, or
  * via the MPD (Media Presentation Description) file in response to a MPD request.

- **Option2: Configuration Exchange as Joining in a SWARM**: In view of the interworking usage of PPSP as a generally accepted suite of protocols to bridging different P2P systems, who may differ in specific choice of hash functions and other parameters, there SHOULD be a way of parameter negotiation mechanism across different systems. Negotiation may also introduce flexibility in a single system. E.g. large files or mobile peers may prefer more compact way of exchanging this information. Therefore, the tracker could include a swarm-specific BF configuration parameters into the OK response to the JOIN request from a new-coming peer (as labeled by (a3) in Figure 3).

5. Security Considerations

TBA

6. IANA Considerations

None.

7. References

7.1. Normative References
7.2. Informative References


[P2P-limit]
Feng, C., Li, B., and B. Li, "Understanding the performance gap between pull-based mesh streaming protocols and fundamental limits", in Proc. of IEEE INFOCOM, 2009.


Authors’ Addresses

Lingli Deng
China Mobile

Email: denglingli@chinamobile.com

Jin Peng
China Mobile

Email: pengjin@chinamobile.com

Yunfei Zhang
CoolPad

Email: hishigh@gmail.com