BUIP093: Graphene v.2 Improvements and Extensions

Date: 21 September, 2018

Proposers: George Bissias (gbiss@cs.umass.edu) and Brian Levine (brian@cs.umass.edu)
edit Sponsored by Andrew Clifford

Motivation

Graphene is a block relay protocol that is capable of transmitting blocks across the Bitcoin Cash network much more compactly than previous solutions such as Xtreme Thinblocks (XThin) and Compact Blocks. We designed and helped code for Bitcoin Unlimited an initial implementation of Graphene, which was released with Bitcoin Unlimited version.

Limitations to Graphene v.1

- Graphene decode failures are not entirely independent among peers.
- Cheap hashes used in the IBLT are susceptible to random or intentional collisions.
- Decode failures can occur quite frequently when peer mempools are out-of-sync.
- Existing failover solutions do not leverage partial information from the failed Graphene block.
- Graphene block optimization and creation could be made less CPU intensive.
- Currently there is no way to send an expedited Graphene block.
- Some fields in the Graphene data structure have been "over-engineered" and could potentially be reduced in size.

Objectives

The project will proceed in two phases: *Improvements* and *Extensions*.

Phase 1: Improvements

The following items will be prototyped as functional patches based on the *dev* branch of the BitcoinUnlimited (BU) software repository. In the event that the BU team agrees that a patch meets satisfactory performance, security, and utility requirements, we will submit it as a pull request (PR) against the *dev* branch. Our team will work with the BU team to refine the PR until the latter team deems that the PR is suitable to be merged or otherwise abandoned.

1. Protocol specification for current IBLT implementation: This is absent from the Graphene v.1 specification and it is unreasonable to expect other teams to implement a fairly new data structure without a specification.
2. Use SipHash plus salt for generating Graphene cheap hashes: By passing the 32-byte transaction ID through a SipHash with a unique salt, cheap hash collisions can be made to occur independently between instantiations of Graphene blocks.
3. Allow arbitrary random seeds to be used for IBLT hash functions: The current IBLT implementation uses a fixed set of *i* seeds for the *i* hash functions. This static setting makes a Graphene block at the same block height more likely to fail to decode for multiple receiving peers.
4. Use filterInventoryKnown Bloom filter: Graphene currently assumes that the receiver's mempool contains all transactions in the block. When this assumption is grossly inaccurate, the block will fail to decode. However, by first passing transactions in the block through the filterInventoryKnown Bloom filter, the sender can estimate what transactions the receiver is likely missing and send them to the receiver.
5. Improve Graphene optimization algorithm: During the creation of a Graphene block, the sender must find the optimal sizes for the IBLT and Bloom filter data structures. In the original Graphene paper we introduce a simple equation for choosing these sizes, but for small blocks it can be somewhat inaccurate. To address this problem,

the current implementation uses a brute-force search, but it would be better to search only over small blocks and use the simple equation for larger blocks.

6. Re-tune Graphene parameters: In research that will be presented in a forthcoming paper, we have identified that for larger blocks, the computation of optimal IBLT settings should account for the natural variance of the Bloom filter. By accounting for this variance, we hope to improve the decode rate of Graphene blocks.
7. Implement Expedited Graphene: Unlike other block formats such as XThin, the sender of a Graphene block must have a good idea about the size of the receiver's mempool in order to properly construct the block, which complicates the process of sending an expedited block. We will explore the use of heuristics to estimate the size of the receiver's mempool and enable expedited Graphene blocks.
8. Use CFastFilter: Graphene places 32-byte transaction IDs into a CBloomFilter object. Each insertion requires that the ID be hashed once for each hash function. In contrast, the CFastFilter implementation avoids hashing by simply using entropy from the ID itself to serve as hash values. It is possible that we can replace our CBloomFilter instance with CFastFilter.
9. Reduce IBLT checksum: The current IBLT implementation uses a 4-byte checksum field to detect decode errors, but this is likely more than is necessary. A preliminary investigation suggests that it may be possible to achieve additional space savings against a minimal increase in failure rate by giving the checksum field a variable value.
10. Update protocol specification for Graphene: We will update the Graphene specification to reflect the changes made above.

Phase 2: Extensions

We propose to extend the Graphene protocol to provide a means for size-efficient mempool synchronization and to allow Graphene to better handle decode failures. The proposed extensions have been carefully evaluated by our team in a forthcoming paper. Distinct from Phase 1, some additional research will likely be necessary to support the development of that implementation. We would like to reserve 25% of the overall time in this phase to writing, executing, and evaluating numerical simulations of associated data structures as well as theoretical analysis useful for improving the safety, performance, or usability of the implementation. Similar to Phase 1, we commit to delivering prototypes as functional patches based on the *dev* branch of the BU software repository. Upon request from the BU team, we will furthermore submit these patches as PRs against the *dev* branch and see them through to the point where they are either merged to *dev* or abandon by the BU team.

Major deliverables

For both mempool synchronization and Graphene Extended we propose to complete the following tasks.

1. Initial specification of the protocol and iteration with BU team and broader community to refine document
2. Any new theoretical or experimental analysis necessary to determine proper constants and settings needed for protocol operation
3. Prototype implementation based off the *dev* branch of the BU codebase
4. Prototype testing and refinement
5. Productize prototype, release code as PR against BU's dev branch, and iterate with BU developers until PR can be merged

Overview of Graphene-based mempool synchronization

In addition to efficient block propagation, Graphene can be used for intermittent mempool synchronization. In our approach, peers will maintain an additional *e* neighbors. Intermittently, from the auxiliary pool of *e* known (and willing) peers on the network, a remote peer is selected randomly to perform a mempool synchronization. The protocol description can be found in the our draft publication.

Overview of Graphene Extended
The current version of Graphene assumes the receiver has all block transactions. However, this is not always the case when mempools are significantly desynchronized. In order to greatly mollify this problem, we will implement **Graphene Extended**, which uses the same core logic as mempool synchronization described above. Thus, with the exception of network messaging, much of the associated code will be reusable. A detailed description of the Graphene Extended protocol can be found in our draft publication.

Project Duration

The total anticipated project duration is 10 months (based on part-time work):

- Phase 1: 4 months total
- Phase 2: 6 months total: 4.5 months for mempool synchronization and 1.5 months for Graphene Extended

Budget

The total requested budget for this project is $50,000, or $5,000 per month of the anticipated project duration.

Phase 1

$1K: Protocol specification for current IBLT implementation
$2K: Use SipHash plus salt for generating Graphene cheap hashes
$1.5K: Allow arbitrary random seeds to be used for IBLT hash functions
$4K: Use filterInventoryKnown to anticipate and mitigate Graphene block decode failures
$1.5K: Improve Graphene optimization algorithm
$2K: Re-tune Graphene parameters based on results from new paper
$3.5K: Implement Expedited Graphene
$2K: Use CFastFilter instead of CBloomFilter
$1.5K: Reduce IBLT checksum
$1K: Update protocol specification for Graphene

Phase 2

Mempool synchronization:
$2K: Protocol specification
$6K: Theoretical and experimental analysis
$8K: Prototype implementation
$4K: Prototype testing and refinement
$2.5K: Productize prototype
Graphene Extended
$1K: Protocol specification
$1.5K: Theoretical and experimental analysis
$2.5: Prototype implementation
$1.5: Prototype testing and refinement
$1K: Productize prototype