P2P Work Spec

The P2P network will have 3 stages of development and research, given the development and business timelines.

Stage 1: A working and integrated P2P network

The goal is to have a working P2P message gossiping network which propagates upper layer messages. At the end of this stage, the P2P network code will be integrated with other components of the Logos network software. This is for testing and demo only. Integration is the key. We will improve the performance in Stage 2.

I think the P2P overlay network itself should be kept and the bitcoin transaction and block etc should be removed. To be exact, I think we should keep the following.
-- Peer discovery: all 5 ways to discover IP address and port of nodes:
-- 1) Address database.
-- 2) User specified via command line argument,
-- 3) DNS seeding,
-- 4) Hard-coded seeds,
-- 5) Discovery via "getaddr" and "addr" network messages.
-- Address manager: managing the IP address and port of nodes discovered. I think we should keep the current logic without much change. In more detail, the address manager will organize the IP/Port in buckets ("new" and "tried"), trace the "last heard from" timestamp, asynchronously dump the addresses to peers.dat should be kept, etc as the current bitcoin P2P network does.
-- Peer connectivity: using the addresses managed by the "address manager" to make (e.g., 8) outbound connections, accepting inbound connections, disconnecting from peers appeared offline.
Naturally all the messages used by above functionalities should be kept. I think we can leave the message format as is for "Stage 1". I think you are more familiar with the code than I do. So please let us know what you think, and especially if there are P2P overlay functionalities (not transaction, nor block related) that you think we should keep.

One functionality that is moved to upper layer is the decision of continue propagate a message or not. Another change is that a node always sends messages (if upper layer decide to propagate) to all of its peers without asking if they need them. We know it is very inefficient, and we can discuss. The current plan is to improve it in Stage 2.

Carl told me you already got your code working, which is great. I also like your API to the upper layer.I think their meaning should be:
*bool PropagateMessage(const void *message, unsigned size);*

// A upper layer logic uses this function to propagate a message to the whole P2P network.
// This function returns false only if the local P2P node has no connections.
*bool ReceiveMessageCallback(const void *message, unsigned size);*
// The callback is provided by the upper layer and called by the P2P layer. It is called whenever a upper layer message is received.
//If the function returns false, then the P2P local node will not propagate this message to its connections. Otherwise, the P2P local node sends the message to all its connections. (As said before, we plan to work on "ask then send" for performance in Stage 2).

You probably already have a way to distinguish the upper layer messages from P2P layer messages. Please let us know how that is done currently.

We also have several **integration** related tasks (require some understanding of the upper layer code and collaboration with the NEW YORK team)
-- Clean the code. Please delete the code that is not going to be used completely instead of commenting it out.
-- Change the database library, if leveldb is used, please replace it with LMDB (which is used in other components of our LOGOS network software).
-- Change hashing functions to Blake2b when hashing is needed.
-- Work with the NEW YORK team to design the ReceiveMessageCallback(message) callback function. This function will (1) verify the message according to the upper layer logic, (2) make sure the same message will be propagated only once. We will discuss the detail later.
-- Use Boost Asio lib for networking as the Logos consensus component does currently. I.e., we will use Boost Asio for sending and receiving data. After a message is received, we have two choices, which Dmitry can decide. The design decision should be presented and discussed before implementation.
1) merge the bitcoin header and our message header. And expand the current message assembler.
2) keep the bitcoin header. Hence a P2P message can wrap a upper layer message. P2P messages will have their own message assembler also. If a message is received, it is unwrapped and send to upper layer.
Note: we understand this is a sizeable change, and most likely will change the thread model. So let’s discuss.
-- Documentation, especially the complicated logics such as the disconnection logic. We should vet them and decide what to keep and what to remove. Dmitry can decide a efficient way of communication. Maybe comments (easily searchable such as //DMITRY:) in the code is a good start. The comments will be discussed and summarized.

Stage 2: Performance improvement

The Goal is to (1) reduce propagation delay of consensus messages sent by the delegates, (2) reduce bandwidth cost of both P2P network nodes and the delegate nodes. We will have to do some research and design. Below is what I think currently.

Method: Currently in Stage 1, all the transactions of a consensus session run by the delegates are sent together with the Post-commit message (which is the confirmation). They consume a large portion of the delegates' bandwidth and very likely cost large propagation delay. Hence we will propagate the single transactions early, not by the delegates, but by the clients (the initiators of the transactions).

Issues and details to work on:
-- a single transaction could be received later than the post-commit of the transaction (a batch of transactions to be exact). We will need to handle this and other kinds of dependencies.
-- efficiently identify single transactions in terms of space, so that messages could be shorter. I think we can use shorter hash values.
-- "ask, then send if needed".
-- quick verify, i.e., the ReceiveMessageCallback(message) callback function will verify the message quickly by skipping some of the steps (which could be time consuming or even incomplete due to missing information). Because the verification is quicker, the message will be propagated earlier and faster, because the message is propagated by the local node only after the ReceiveMessageCallback returns. Of course, the skipped steps (very unlikely to fail) will be performed later.

Stage 3 P2P network internal improvement (only if needed):

After stage 2, only if needed, we might have to work on P2P network internals, e.g., to reduce the network diameter, to reduce per hop delay, or improve robustness against attacks such as sybil attack and eclipse attack, etc.