P2p Backup Consensus

Overview

Preferred way of exchanging consensus messages between the delegates in the logos core software is direct TCP/IP connection. When consensus protocol does not reach two-third majority, a primary delegate enables P2p subsystem for a consensus message propagation in addition to the TCP/IP connection. When a backup delegate receives the consensus message from P2p subsystem, it responds to the primary delegate via both TCP/IP and P2p subsystem. The primary delegate keeps on using TCP/IP and P2p to exchange consensus messages until it deduces that P2p subsystem is no longer needed for the consensus message propagation. When the backup delegate stops receiving consensus messages via P2p subsystem, it discontinues propagating consensus messages via P2p subsystem.

Class Diagram

P2p backup and direct consensus represent a typical Producer-Consumer paradigm. Where producers in the logos architecture are ConsensusNetIO::OnData() and ConsensusContainer::OnP2pReceive(), which receive consensus messages respectively from TCP/IP and P2p. Consumer of the consensus messages is BackupDelegate with DelegateBridge being an intermediary between BackupDelegate and receiving channels.

P2p backup consensus support extends existing classes and implements new classes:
- ConsensusP2pBridge abstracts P2p layer from ConsensusManager and ConsensusNetIO. It broadcasts (Broadcast()) Post_Committed_Block message and sends (SendP2p()) consensus messages if p2p is enabled. The virtual OnTimeout() function has to be implemented by the subclass and handle _enable_p2p flag reset. ConsensusManager and ConsensusNetIO are subclassing ConsensusP2pBridge.
- ConsensusMsgConsumer is an abstract class and requires the subclasses implement message parsing (Parse()) and consuming (OnMessage()). ConsensusMsgSink is subclassing ConsensusMsgConsumer and DelegateBridge implements these methods.
- ConsensusMsgProducer is an abstract class and requires the subclasses to implement AddToConsensusQueue(). It dictates common interface of adding consensus messages to the consensus queue. ConsensusManager and ConsensusNetio are subclassing ConsensusMsgProducer.
- MessageBase is an abstract class with no members. MessagePrequel is subclassing MessageBase. This is required so that heterogeneous consensus message can be stored in one queue.
- ConsensusMsgSink implements the message queue and is at the same time the consumer of the queue. ConsensusManager and ConsensusNetIO add consensus messages via ConsensusMsgProducer interface (AddToConsensusQueue()) which in turn calls Push(). Push() handles the queue locking and if the queue is not currently being consumed then posts ConsensusMsgConsumer::OnMessage() to the boost thread-pool via Post(). As long as the queue is not empty it is continuously asynchronously consumed via Post() and Pop() by requesting a new thread from the thread-pool for each new message. The queue stores Message instances. Message is a tupple of message type, and the pointer to MessageBase. Message is a private type declared within ConsensusMsgSink class scope. Message’s message_type allows dowcasting from MessageBase to the correct consensus message type object. _msg_cnt maintains the number of received direct and p2p messages. DelegateBridge is subclassing ConsensusMsgSink.

The class diagram is shown on Figure 1.

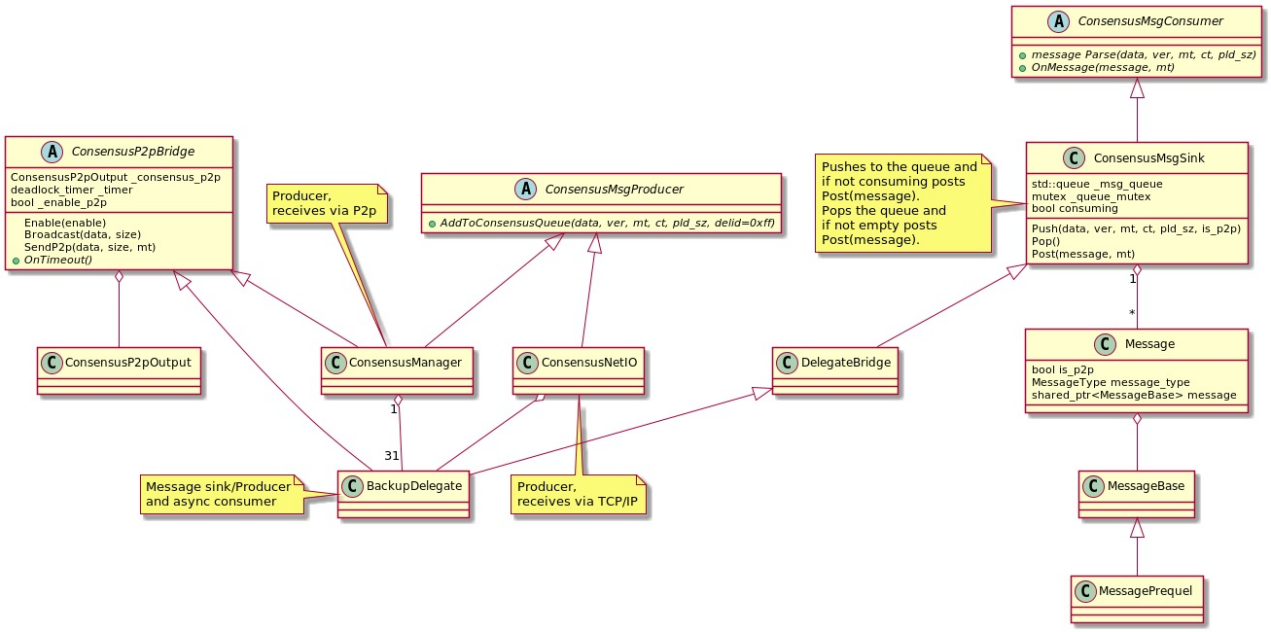

Figure 1. P2p backup consensus class diagram.

Sequence Diagrams

Enabling P2p backup consensus by the primary and backup delegates

The diagrams are shown on Figure 2 and 3.

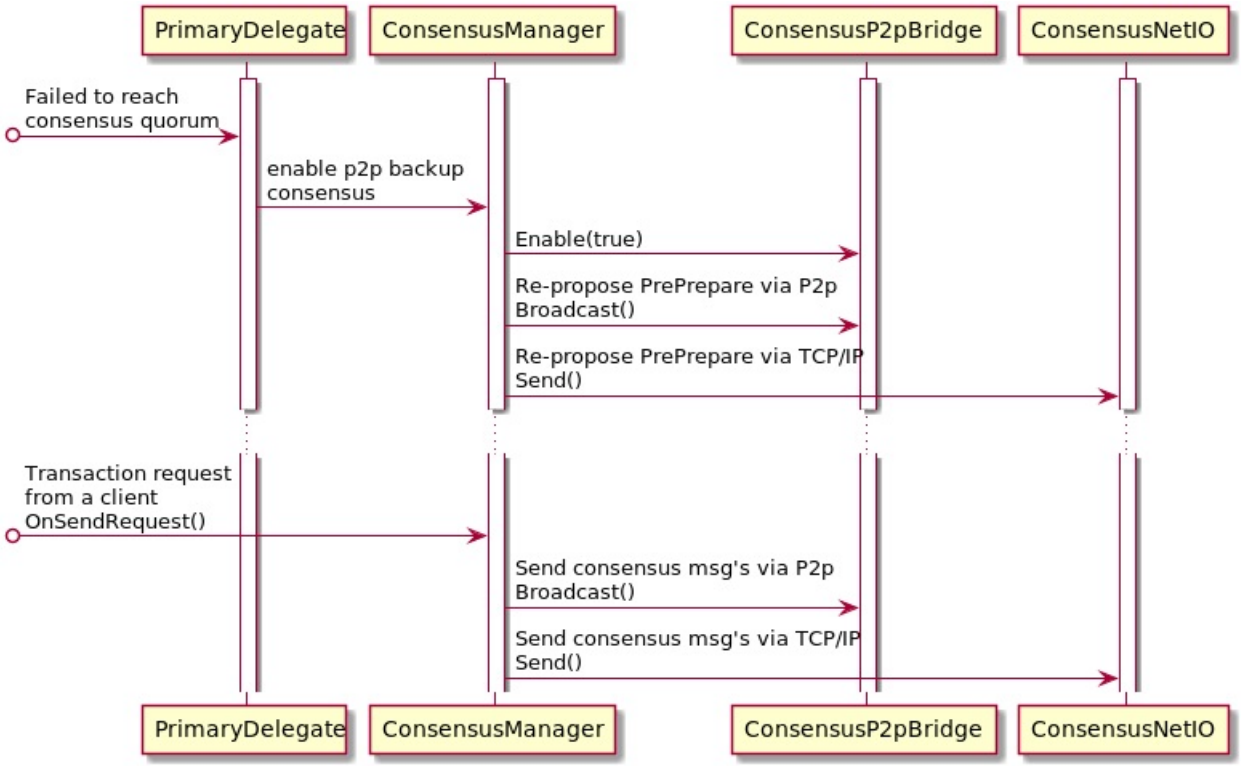

Figure 2. P2p enabling on the primary delegate

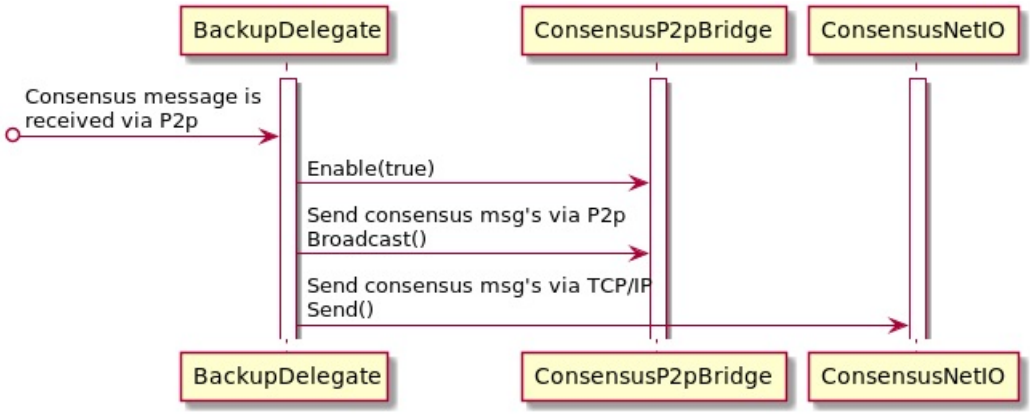

Figure 3. P2p enabling on the backup delegate.

When the primary delegate fails to reach the consensus quorum, it notifies ConsensusManager, which in turn enables P2p backup consensus via ConsensusP2pBridge. It then re-proposes PrePrepare message vi P2p subsystem (ConsensusP2pBridge::Broadcast()) and direct connection (ConsensusNetIO::Send(), via BackupDelegate::Send()). When a transaction request is received by ConsensusManager, it broadcasts the request via P2p subsystem and direct connection to the delegates. Both communication channels are used for all consensus messages. The backup delegate P2p backup consensus is enabled when the backup delegate receives a consensus message from the primary delegate via P2p. Once P2p is enabled, the backup delegate replies to the primary delegate via both P2p (ConsensusP2pBridge::Broadcast()) and direct connection to the primary delegate (ConsensusNetIO::Send()).

Producing P2p consensus message

The diagram is shown on Figure 4.

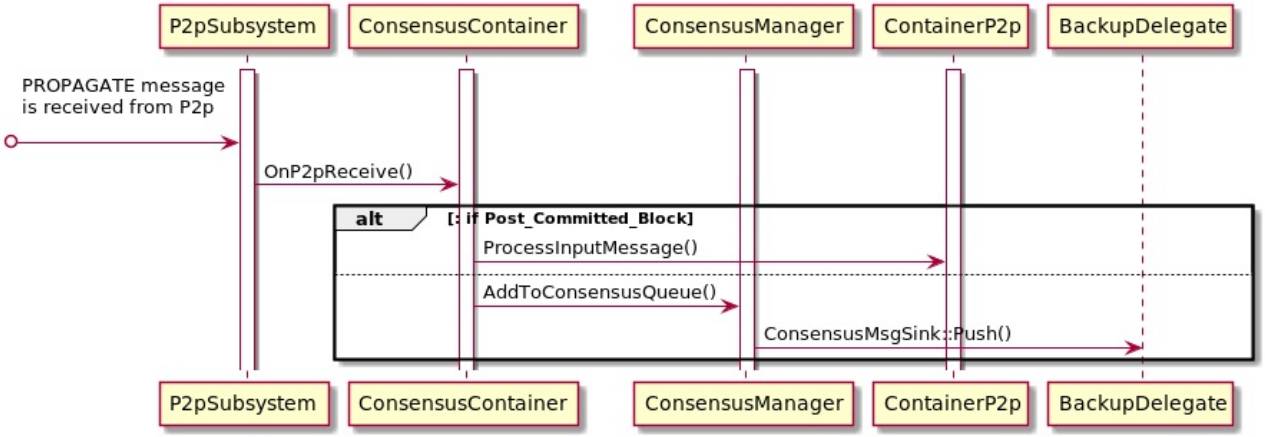

Figure 4. Producing P2p consensus message.

There are a number of classes that are part of the P2p subsystem and which are providing propagation of the received message to the consensus layer. The collection of this classes are denoted on the figure by P2pSubsystem. P2p subsystem delivers received PROPAGATE message to ConsensusContainer (OnP2pReceive()). If the received message is Post_Committed_Block then ConsensusContainer calls ContainerP2p::ProcessInputMessage(), which validates the message and saves it to the database. For any other type of consensus messages ConsensusContainer calls ConsensusManager::AddToConsensusQueue, which in turns calls ConsensusMsgSink::Push() via BackupDelegate. Push() inserts the message into the message queue and may also consume the front of the queue.

Producing TCP/IP consensus message

The diagram is shown on Figure 5.

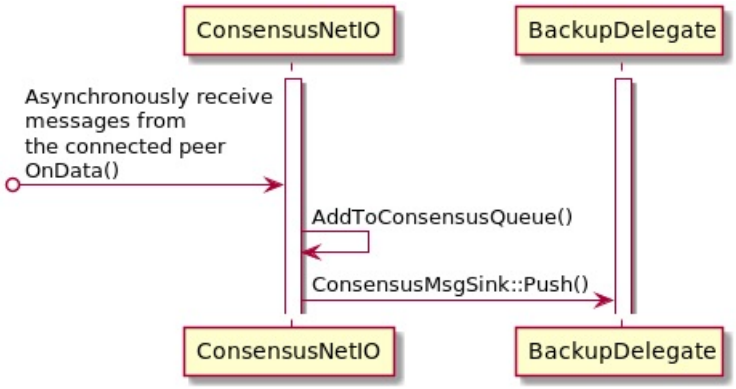

Figure 5. Producing TCP/IP consensus message

Direct connection message are received by ConsensusNetIO::OnData() from connected peer via TCP/IP socket. Consensus messages are added to the message queue via calls to AddConsensusQueue() and ConsensusMsgSink::Push() via BackupDelegate. Push() inserts the message into the message queue and may also consume the front of the queue.

Consuming of consensus messages

The diagram is shown on Figure 6.

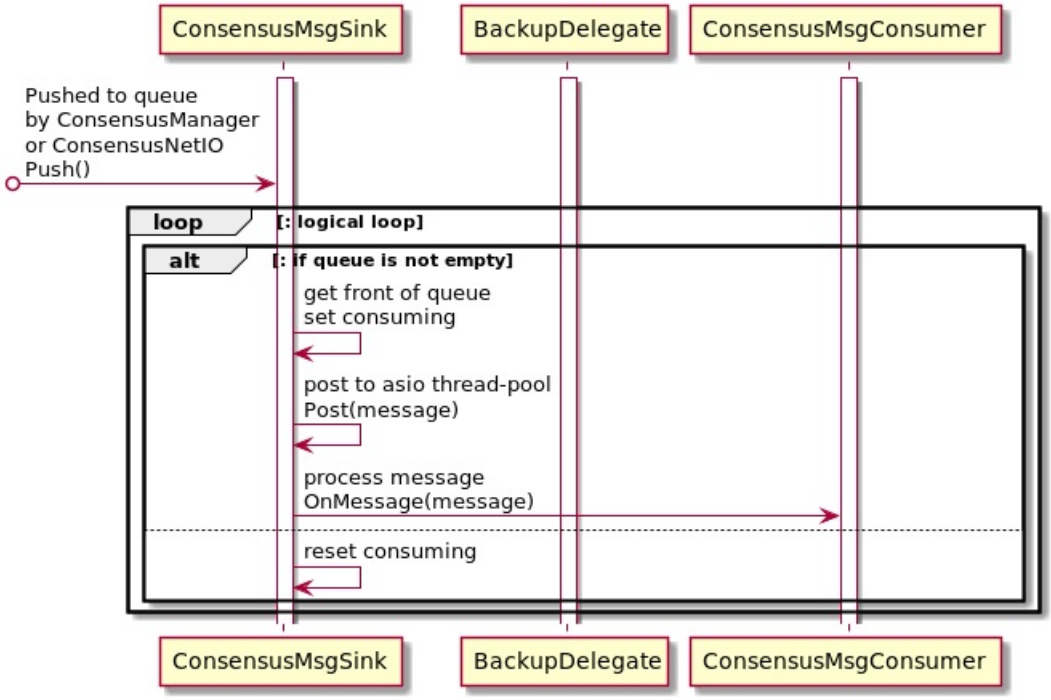

Figure 6. Consuming of consensus message

Consensus message is pushed to the message queue by either ConsensusManager when a message is received via P2p subsystem or by ConsensusNetIO when a message is received via TCP/IP. If the message queue is not currently being consumed (consuming is false) then the queue is popped and the message is posted (Post()) to the asio thread pool to be processed by

ConsensusMsgConsumer::OnMessage() (implemented by BackupDelegate). If the queue is not empty after the message is processed, then the process is repeated; i.e. pop the queue, post the message to the thread-pool to OnMessage(). This flow doesn’t use one dedicated thread to consume the queue. Instead it requests a thread from the thread-pool each time a new message is popped from the queue providing fair distribution of the thread resource between the backup delegates.

Disabling P2p backup consensus by the primary and backup delegates

The diagrams are shown on Figure 7 and 8.

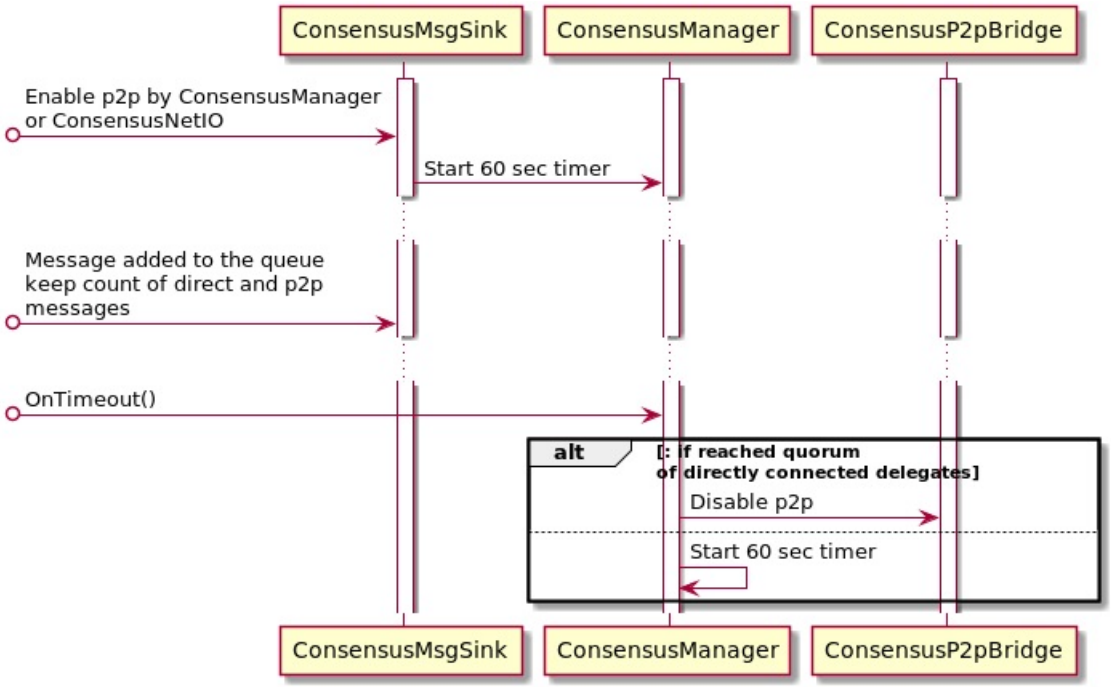

Figure 7. P2p disabling on the primary delegate.

Once P2p is enabled on the primary delegate, a 60 seconds timer is started. A counter is maintained of the consensus messages received via P2p and TCP/IP. When the timer times out, ConsensusManager analyses message count on all backup delegates. If the weighted count of messages received via TCP/IP reaches 2/3 quorum then P2p subsystem is disabled. Otherwise P2p remains enabled and another 60 sec timer is started.

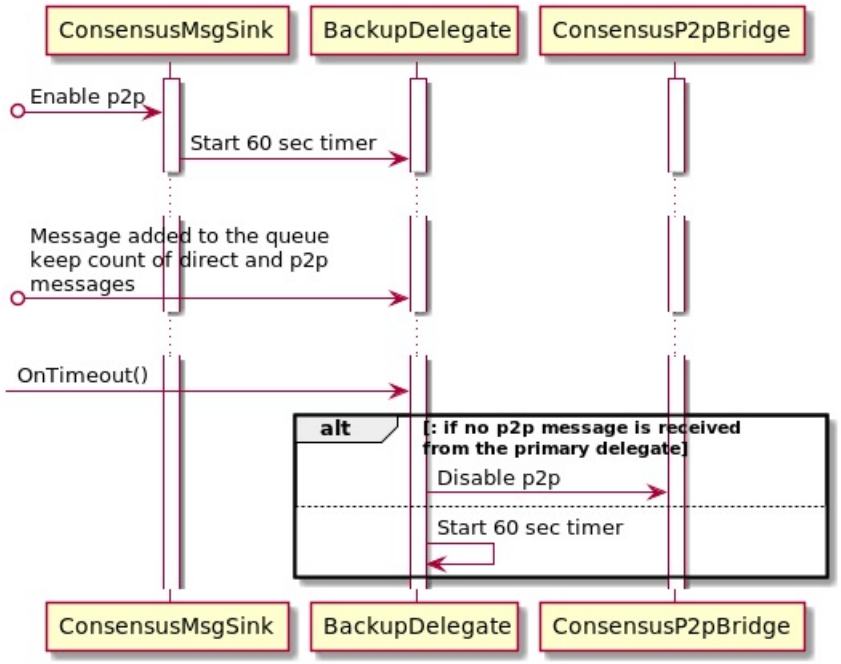

Figure 8. P2p disabling on the backup delegate.

Once P2p is enabled on the backup delegate, a 60 seconds timer is started. A counter is maintained of the consensus messages received via P2p and TCP/IP. When the timer times out, if no p2p messages were received, then P2p subsystem is disabled. Otherwise P2p remains enabled and another 60 sec timer is started.

Protocol

P2p consensus message is layered on top of P2p PROPAGATE message. P2p broadcasts messages to all delegates. This means that some delegates will receive messages that should not be directred to them. While the delegate can correctly identify these type of messages and discard them, it’ll use the delegates resources. This could be addressed at the level bellow consensus protocol. mpf field of MessagePrequel can be used to add destination delegate id and epoch number which allow delivering the message to the correct delegate.