horizontal line

Notes of Timeout values

We list the timeout values in the consensus requirements and epoch requirements documents.

# summary

|  |  |
| --- | --- |
| Timeout Name | Timeout Value (seconds) |
| Client re-submission | 15 |
| Primary pre-prepare | 60 |
| Primary post-prepare | 60 |
| Microblock | random\_timeout(60, 60) |
| Epochblock | random\_timeout(60, 60) |
| Post-prepared but not post-committed BSB rerun | random\_timeout(60, 60) |
| Client requests sent to non-default primary delegates | 5  If no msg from primary in 60 seconds, 1 consensus round time, e.g. 3  Else (Client re-submission timeout)/2, e.g. 15 |
| Fallback consensus Via secondary waiting list | random\_timeout(10, 20)  random\_timeout(20,20) during epoch transition (-20 to 0). Due to the design of the EpochManager, When a new EpochManager starts to work at -20, it could become a backup delegate for a BSB. After preparing for it, the BSB will be inserted in the 2nd waitingList. With a larger timeout value, it guarantees that the new EpochManager will be the one to re-propose the BSB if needed. |
| Recall | 300 |

# 

# 

# Client re-submission timeout

When a client does not receive the reply of its request within the re-submission timeout, it will re-submit the request to another delegate.

* A small timeout value is good from the client’s point of view, when the primary is faulty (dead or becomes slow due to local software or hardware issues, or malicious), or the message is lost.
* Otherwise, a small timeout value is counterproductive to the whole network. Our network is load balanced. I.e., delegates should have the same amount of transactions sent to them. If a client does not receive a reply simply because the delegate is busy, then re-submission only make things worse since all other delegates are busy too.
* As a simple client, it may not have the statistical information about recent confirmation time. So I suggest a constant timeout value, say 30 seconds.

# Primary pre-prepare timeout

After sending out the pre-prepare message, the primary sets the pre-prepare timer. When it expires, the primary will go to the recall\_wait state.

* A small timeout value seems to be good from the client’s point of view because the client can retry quicker after receiving the reject message (if we decide to send a reject). However, if a valid request is proposed by a honest primary and the primary cannot finish the consensus session, then re-submitting the request to other delegates will not help, most likely.
* Since a small timeout value does not benefit anyone, I suggest to use a large constant value, such as 60 seconds.

# Primary post-prepare timeout

After sending out the post-prepare message, the primary sets the post-prepare timer. When it expires, the primary will go to the recall\_wait state.

* For the same reason as primary pre-prepare timeout, I suggest to use a large constant value, such as 60 seconds.

# 

# A naïve random\_timeout() function

I think the rest of timeout values need to be randomized so that a small number of timeout values are small and a large number of timeout values are large, to reduce redundant and concurrent sessions. So I propose a simple function as below.

#define NUM\_DELEGATES 32

int random\_timeout(int init\_delay, int range)

{

int offset;

int x = rand() % NUM\_DELEGATES;

if (x < 2) offset = 0;

else if (x < 4) offset = range/2;

else offset = range;

return init\_delay + offset;

}

When the delegates run: tov = random\_timeout(30, 30); then on average 3 delegates should have tov=30, another 3 delegates should have tov=45, and the rest have tov=60.

# Timeout for Microblock/Epochblock/Post-prepared but not post-committed BSB rerun

A delegate that is not the default primary of these requests will put them in its secondary waiting list with a timer. When the timer expires, the delegate promotes the requests to the primary waiting list and then propose pre-prepares for them soon.

* If two concurrent consensus sessions have overlapping requests in their pre-prepare message, we consider they are redundant. Redundant sessions hurts network throughput, hence reducing redundant and concurrent sessions is the main goal.
* Of course, a microblock should be finished (with fallback) before the next microblock.
* I suggest we use random\_timeout(60, 60);

# 

# Timeout of client requests sent to non-default primary delegates

When a delegate that is not the default primary receives a request, it inserts the request in the secondary waiting list with a timer. When the timer expires, the delegate promotes the requests to the primary waiting list.

* When the delegate receives the request, the client might have sent the request to other nodes already. From network throughput point of view, to reduce redundant and concurrent sessions, it makes sense to use a randomized and larger timeout value.
* On the other hand, the delegate should pre-prepare the request quick enough so that the client does not timeout and send the request to more delegates, which could result in more redundant sessions.
* Since we allow a client sends a request to multiple delegates, to get timely reply of the request, the client might always send the request to multiple delegates. How to disincentivize?
  + **PoW includes the delegate information, so that the client must recompute the PoW when it sends the request to a different delegate.**
  + And, have a large enough timeout value so that as long as the primary is not faulty, it post-commits before other delegates pre-prepare.
* We cannot simply use the random\_timeout() function because we don’t know how many delegates received the client request.
* This timer seems trickier than others. Do we need an adaptive way to compute the timeout value? Maybe we can borrow the algorithm from the tcp retransmission timeout, which is designed to balance between “timely retransmission of packets indeed lost” and false positives. It was also designed to quickly adapt to network delay changes. For us to use such a algorithm, we also need to measure the message RTTs, which is not difficult.
* I suggest that a delegate keeps track when it received the last message from the default primary. If the delegate has not received messages from the primary for awhile (60 seconds), then use a short timeout (about one consensus session time, 3 seconds), otherwise use a large timeout (half of client re-submission timeout, 15 seconds). Note that delegates could be quiet and don’t talk to each other (since we don’t have a heartbeat), hence resulting in short timeout values which could lead to redundant sessions. I think this is OK since the network is very much underload.

# 

# 

# Timeout of fallback consensus Via secondary waiting list

Receiving a valid pre-prepare (or a valid post-prepare), a backup inserts it in its secondary waiting list with a timer as a alternative of the fallback consensus protocol. When the timer expires, the delegate promotes the batch of requests to the primary waiting list.

* Since all honest backup delegates inserted it in the secondary waiting list, we must reduce redundant sessions.
* I suggest we use random\_timeout(10, 20);