Semi-sync Group Ack

Problem

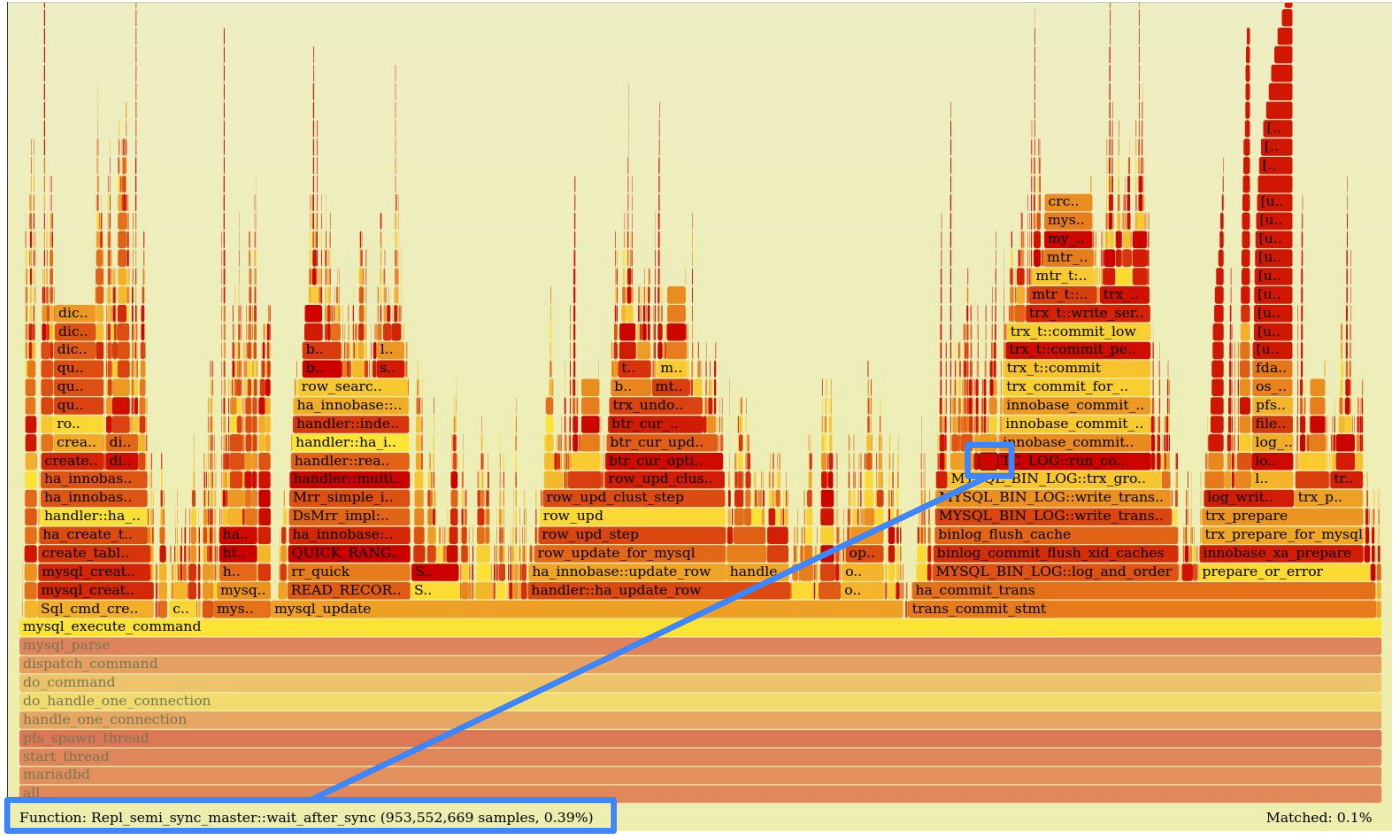
All transactions wait on one cond var

With rpl_semi_sync_master_wait_point = AFTER_COMMIT

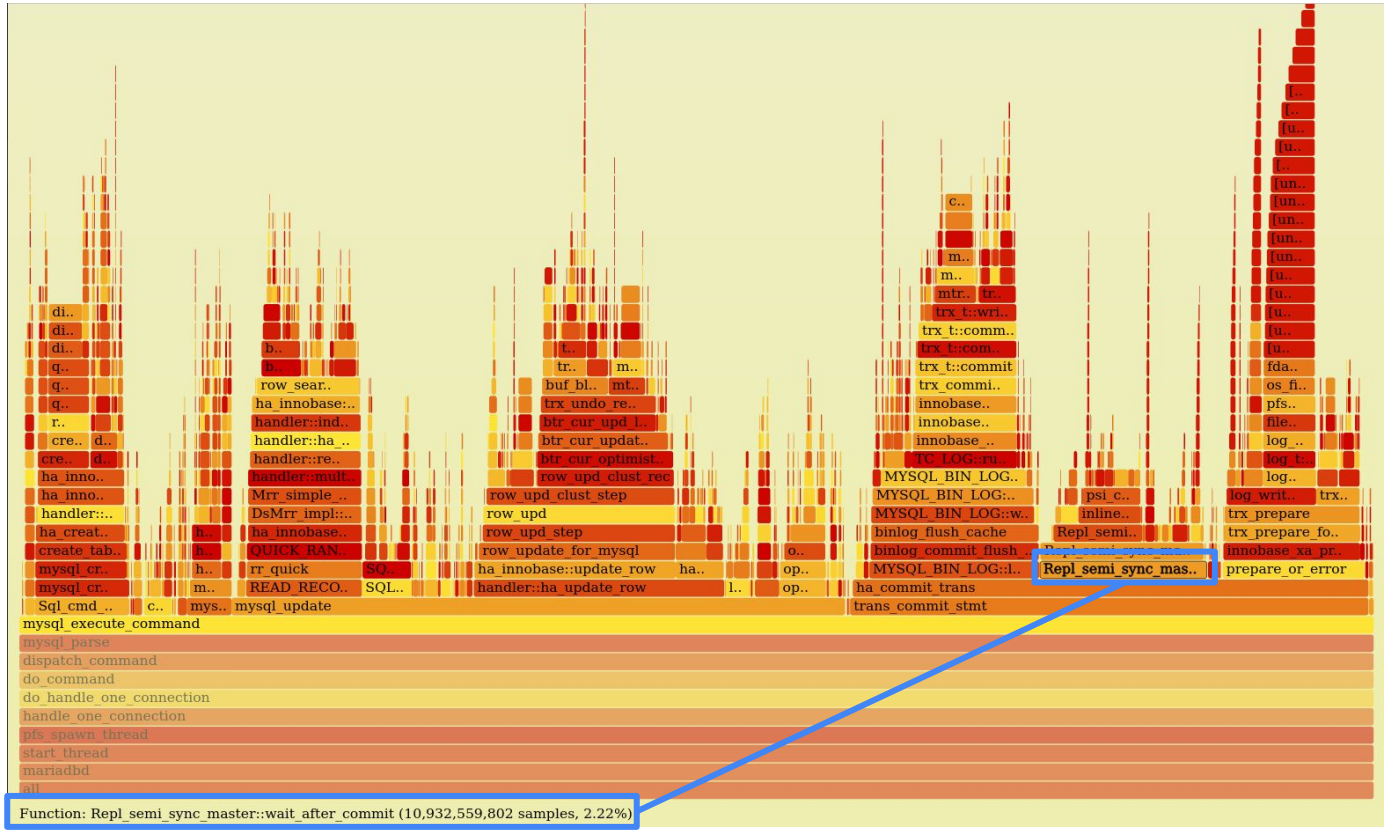
All concurrent transactions are signalled

Flamegraphs

rpl_semi_sync_master_wait_point = AFTER_SYNC

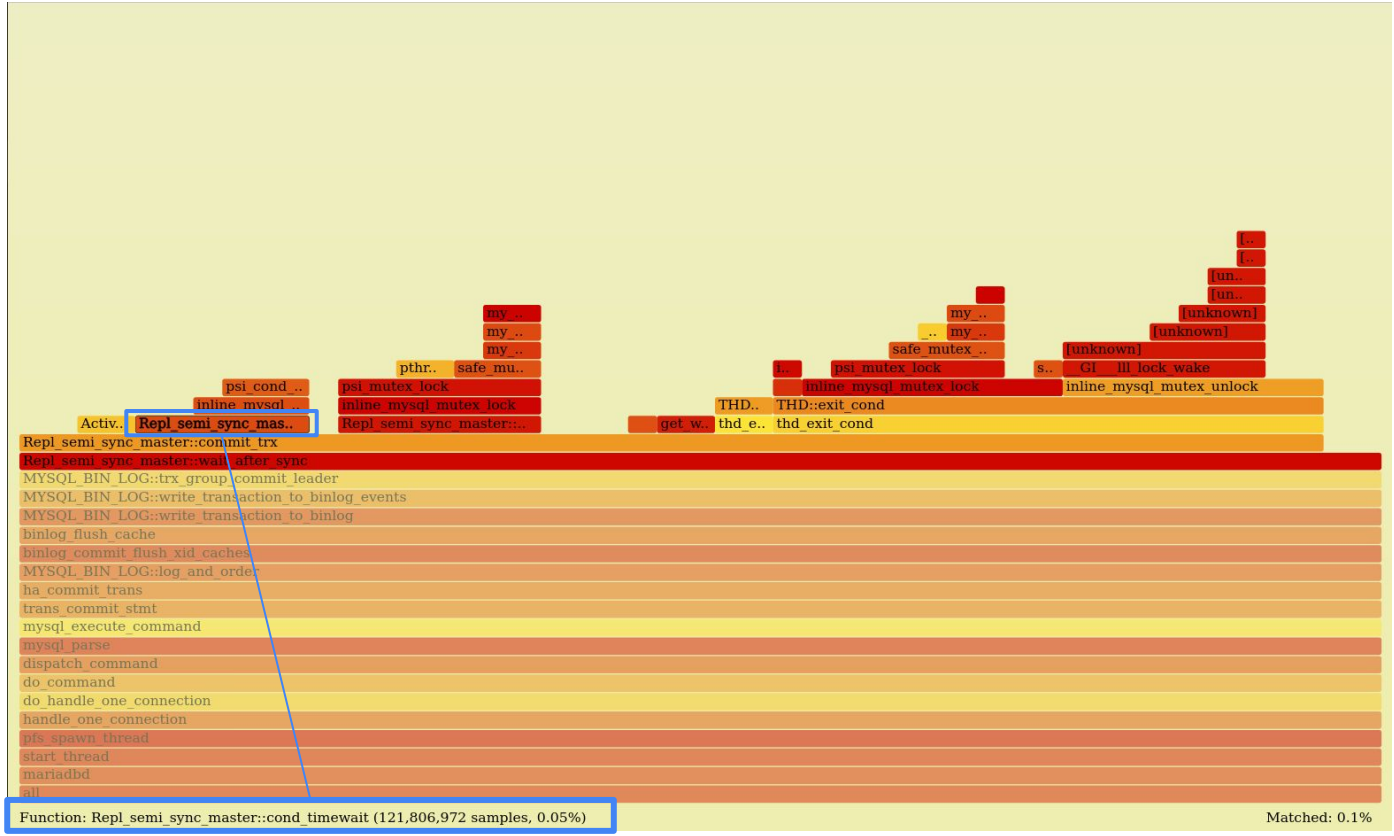


rpl_semi_sync_master_wait_point = AFTER_COMMIT



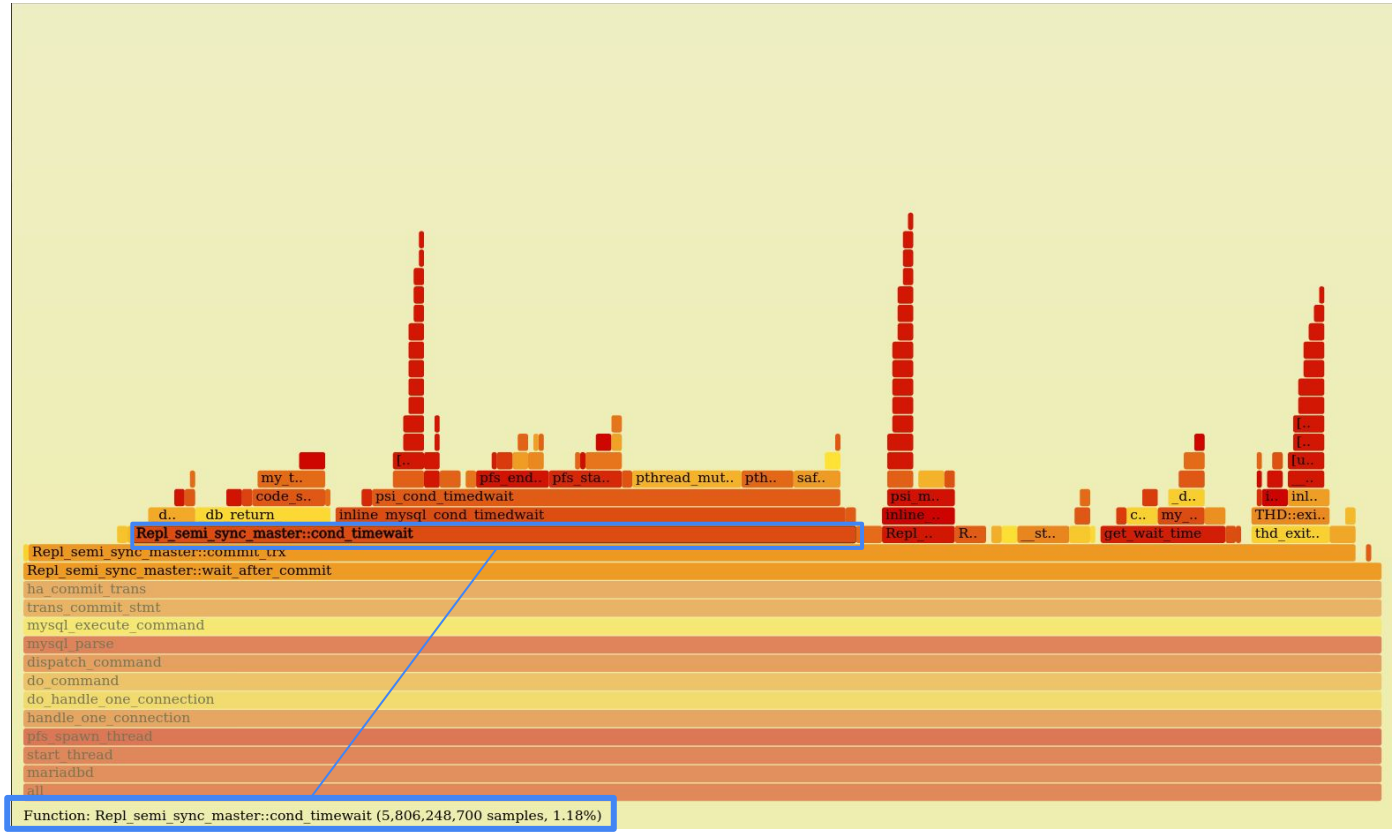
Zooming into commit_trx()

AFTER_SYNC time in cond_timewait



AFTER_SYNC
100k samples

AFTER_COMMIT time in cond_timewait



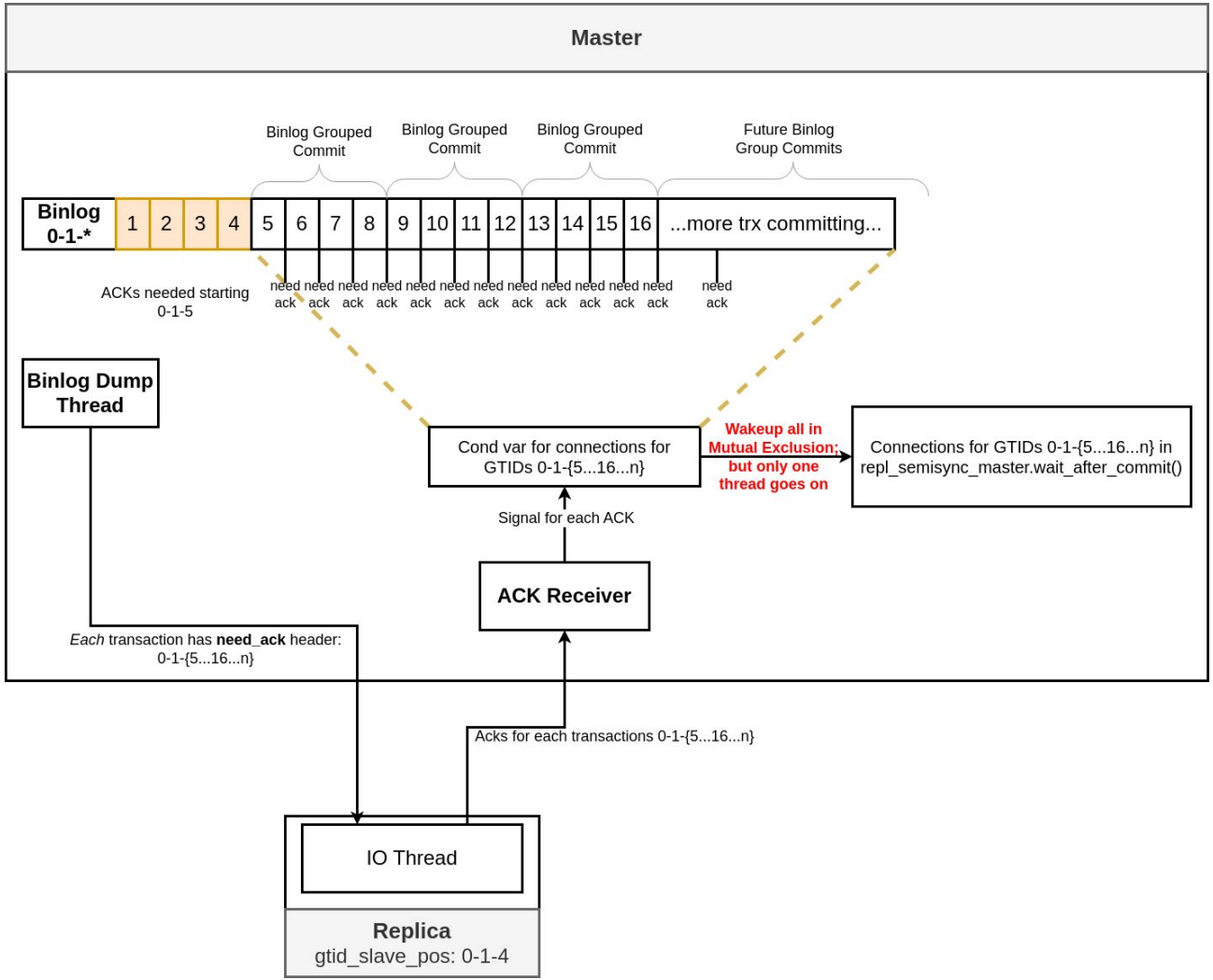
AFTER_SYNC
100k samples

AFTER_COMMIT
5.8B samples

AFTER_COMMIT Complications

Multiple binlog group commits can happen ahead of current waiting trx

Example


Master
Binlog Grouped Commit
Binlog Grouped Commit
Binlog Grouped Commit
Future Binlog Group Commits
Binlog 0-1-*
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ...more trx committing...
ACKs needed starting 0-1-5
need ack
Binlog Dump Thread
Cond var for connections for GTIDs 0-1-{5...16...n}
Wakeup all in Mutual Exclusion; but only one thread goes on
Connections for GTIDs 0-1-{5...16...n} in repl_semisync_master.wait_after_commit()
Signal for each ACK
ACK Receiver
Each transaction has need_ack header: 0-1-{5...16...n}
Acks for each transactions 0-1-{5...16...n}
IO Thread
Replica
gtid_slave_pos: 0-1-4

Proposed Solution

At binlogging time, transactions register themselves with repl_semisync_master in binlog order (but don't yet wait)

..then **after** storage engine commit..

repl_semisync_master waits on just the oldest registered transactions (can be individual or groups)

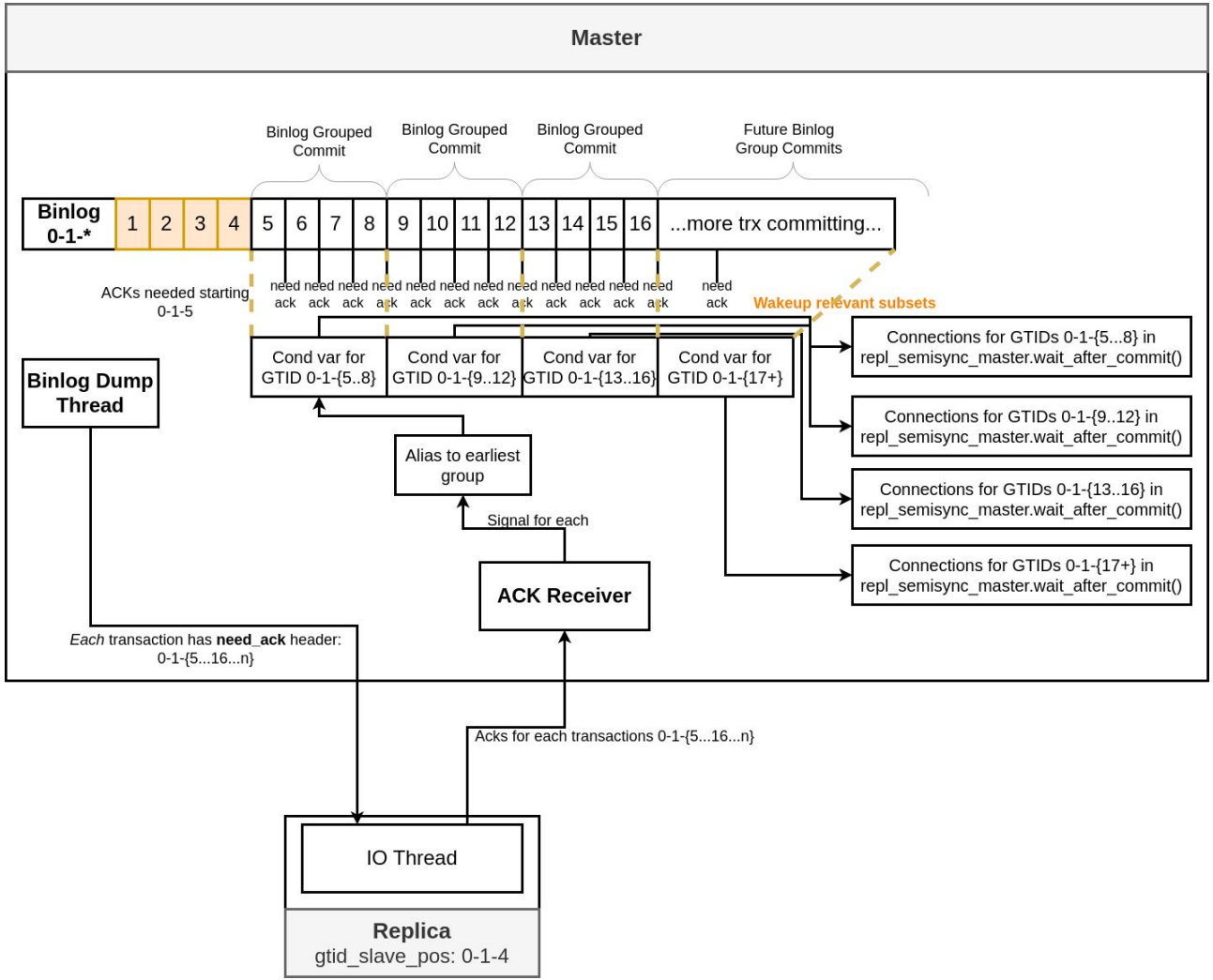
Proposed Solution

Many options here:

1. 1 Cond Var per Registered transactions
    a. No context switching problem, most memory and overhead (more acks from slave)
    b. Can be GA
2. 1 Cond Var for a group of transactions
    a. Limited context switching (only within group), slightly less memory footprint
    b. Can align with binlog group commit boundary
    c. Can be GA
3. 1 Cond Var for a group of transactions + group ack
    a. No context switching, least overhead
    b. New feature I imagine

Option 2 Would Look Something Like…

Option 2


Master
Binlog Grouped Commit
Binlog Grouped Commit
Binlog Grouped Commit
Future Binlog Group Commits
Binlog 0-1-*
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
...more trx committing...
ACKs needed starting 0-1-5
need ack
Wakeup relevant subsets
Cond var for GTID 0-1-{5..8}
Cond var for GTID 0-1-{9..12}
Cond var for GTID 0-1-{13..16}
Cond var for GTID 0-1-{17+}
Connections for GTIDs 0-1-{5...8} in repl_semisync_master.wait_after_commit()
Connections for GTIDs 0-1-{9..12} in repl_semisync_master.wait_after_commit()
Connections for GTIDs 0-1-{13..16} in repl_semisync_master.wait_after_commit()
Connections for GTIDs 0-1-{17+} in repl_semisync_master.wait_after_commit()
Binlog Dump Thread
Alias to earliest group
Signal for each
ACK Receiver
Each transaction has need_ack header: 0-1-{5...16...n}
Acks for each transactions 0-1-{5...16...n}
IO Thread
Replica
gtid_slave_pos: 0-1-4

Option 1 would just be one cond_var per binlog trx

Conclusion After Discussion…

1. In 10.6, do option 1: One condition variable per connection thread
    a. Can reuse THD::COND_wakeup_ready
    b. MDEV-33551
2. in 11.X (new feature), implement group ack
    a. MDEV-33491

Link to Zulip discussion