SANDTABLE BUGS 描述文档

1. PySyncObj

PySyncObj 是一个基于 Python 语言实现的 Raft 协议库，使用 TCP socket 进行网络消息传输，只允许网络分区的网络错误类型。

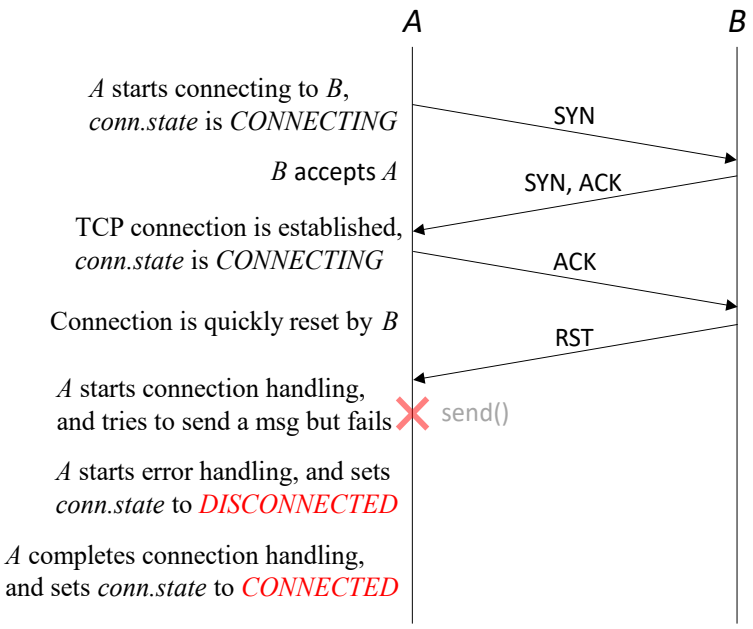
1.1. PySyncObj#161[①]

这是一个存在于 PySyncObj 网络连接建立过程中的代码级 bug，导致代码抛出异常。

PySyncObj 使用 TCP 来作为网络传输层，在获取 socket 文件描述符后，将文件描述符设置为非阻塞。非阻塞的 connect()系统调用立即返回，代码中设置连接状态变量为 CONNECTING，并在事件循环中通过 I/O 多路复用（例如 select()和 poll()）获得 connect()的连接结果，并调用相应的回调函数进行处理，处理完成时，设置连接状态为 CONNECTED。

连接建立的回调函数会调用_sendSelfAddress()将自己的地址发送给对方，而 send()如果返回失败，则会进一步调用 disconnect()函数来关闭连接并释放资源。当_sendSelfAddress()返回后，连接可能已被关闭，且连接状态被设为 DISCONNECTED。但 PySyncObj 没有进行检查连接状态，而直接将连接状态设置为 CONNECTED。这种情况下，该连接的 socket 是空值，下一次试图使用 send()发送数据会抛出 AttributeError 异常，并造成 raft 线程停止运行。

```
if self.__state == CONNECTION_STATE.CONNECTING:
    if self.__onConnected is not None:
        self.__onConnected() # 调用_sendSelfAddress()可能断开连接
    self.__state = CONNECTION_STATE.CONNECTED
```



[①] "Fix disconnection when connecting by tangruize · Pull Request #161 · bakwc/PySyncObj," *GitHub*. https://github.com/bakwc/PySyncObj/pull/161 (accessed May 28, 2023).

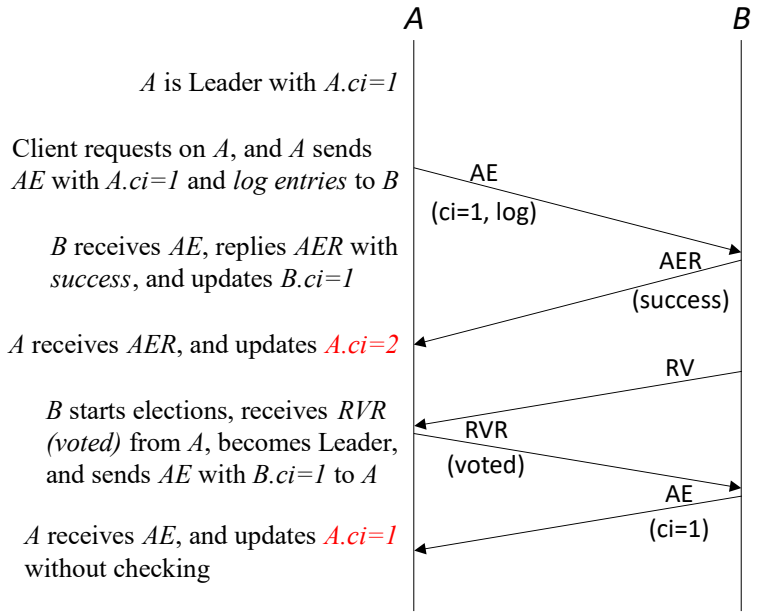
1.2. PySyncObj#166[①]

这是一个协议级 bug，存在某种情况使得 commit index（图中简称 ci）变小，违反了 Raft 协议的安全属性。

根本原因：Follower 接收到来自 Leader 的 AppendEntries 消息（图中简称 AE）后，在消息处理成功的情况下，会将当前的 commit index 设置为 min(leaderCommitIndex, currentLogIndex)，即 Leader 消息中的包含的 commit index 和当前日志项的 index 中的最小值。而对 commit index 的赋值前没有检查 commit index 和 min(leaderCommitIndex, currentLogIndex)的大小，存在后者较小的情况，使得 commit index 不单调递增。

最短触发路径：

1. 当前 Leader 为 A。A 和 B 的 commit index 均为 1
2. A 收到 client 请求，在添加请求中的 log entry 到 log 后，发送 AppendEntries 到 B，AppendEntries 中包含这个请求的 log entry 和当前的 commit index（值为 1）
3. B 收到 AE，处理成功后，返回 AppendEntriesResponse 给 A
4. A 收到 AppendEntriesResponse，更新 current index 为 2
5. 由于某种原因（例如网络延迟等），B 选举超时，发送 ReqeustVote 给 A
6. A 对 B 投票，并返回 ReqeustVoteResponse
7. B 收到 ReqeustVoteResponse，成为 Leader，并发送心跳 AppendEntries 给 A，AppendEntries 中包含当前的 commit index（值为 1）
8. A 收到 AppendEntries，更新 commit index 为 1，而更新前，A 的 commit index 为 2



[①] "Raft commit index is not monotonic · Issue #166 · bakwc/PySyncObj," *GitHub*. https://github.com/bakwc/PySyncObj/issues/166 (accessed May 28, 2023).

```
    if message['type'] == 'append_entries' and message['term']
    >= self.__raftCurrentTerm:
        # 未检查大小
        self.__raftCommitIndex =
    min(leaderCommitIndex, self.__getCurrentLogIndex())
```

1.3. PySyncObj#167-1[①]

这是一个协议级 bug，存在某种情况使得 Leader 中的 next index（图中简称 ni）不大于 match index（图中简称 mi）。违反了 Raft 协议的安全属性，导致已经 match 的 log entries 被不必要地重新发送。此 bug 不会直接造成严重后果，但可能降低性能，并存在潜在的风险。

根本原因：Leader 收到 Follower 回复的 AppendEntriesResponse 消息（PySyncObj 中称为 next_node_idx 消息）后，如果结果为成功，则更新 match index，而不更新 next index。更新后的 match index 可能大于或等于 next index。

```
    if self.__raftState == _RAFT_STATE.LEADER:
        if message['type'] == 'next_node_idx':
            reset = message['reset']
            nextNodeIdx = message['next_node_idx']
            success = message['success']
            currentNodeIdx = nextNodeIdx - 1
            if reset:
                self.__raftNextIndex[node] = nextNodeIdx
            if success:
                # 未同时设置 next index
                self.__raftMatchIndex[node] = currentNodeIdx
```

前提条件：PySyncObj 对原始 Raft 协议做了优化，Leader 在发送 AppendEntries 消息时，自动更新 next index 为当前最大 log entry 的 index 加 1。如果前面有一个 AppendEntries 消息没有被 Follower 收到，后续的 AppendEntries 消息中的 prev log index 就无法与 Follower 的 log 匹配。这种情况下 Follower 返回表示失败的 AppendEntriesResponse 消息，并加上 next node index 来提示 Leader 下一次应该从什么地方发送 log entries。

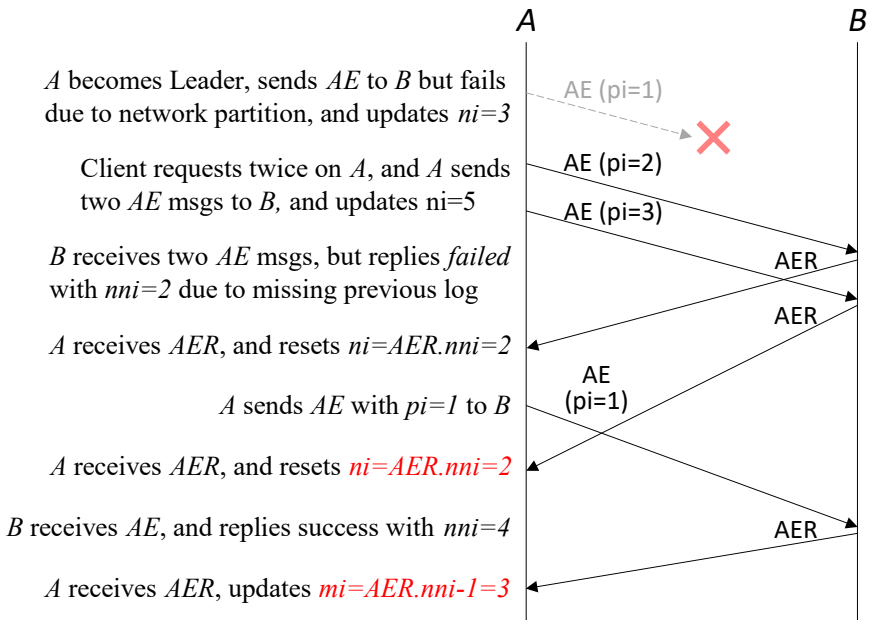
```
    def __sendAppendEntries(self):
        if nextNodeIndex <= self.__getCurrentLogIndex():
            entries = self.__getEntries(nextNodeIndex,
    None, batchSizeBytes)
            # 优化：发送 AppendEntries 修改了 next index
            self.__raftNextIndex[node] = entries[-1][1] + 1
```

最短触发路径：

1. A 和 B 初始化为 Follower，log 中仅包含一个 no-op 的 log entry。

[①] "Raft match index is not monotonic · Issue #167 · bakwc/PySyncObj." https://github.com/bakwc/PySyncObj/issues/167 (accessed May 28, 2023).

2. A 选举成为 Leader，添加一个 no-op 的 log entry 到 log 中，并发送 prev log index 为 1（图中简写为 pi）的 AppendEntries 给 B。发送后，A 对 B 记录的 next index 更新为 3。但由于网络分区，这条 AppendEntries 消息丢失。随后网络恢复。
3. A 接收两条 Client 的 request，分别发送 prev log index 为 2 和 3 的两条 AppendEntries 消息给 B，执行结束后 next index 为 5。
4. B 处理两条 AppendEntries 消息，但由于当前 log 中只有 1 个 log entry，而 prev log index 分别为 2 和 3，因确实日志项返回失败，并在 AppendEntriesResponse 包含 next node index=2。
5. A 收到 AppendEntriesResponse 并将 next index 设置为 2。
6. A（broadcast time out）发送 AppendEntries（prev log index 为 1）给 B。
7. A 收到 AppendEntriesResponse 并将 next index 设置为 2。
8. B 收到 AppendEntries，log 匹配，返回成功，AppendEntriesResponse 消息中 next node index 为 4。
9. A 收到 AppendEntriesResponse，更新 match index 为 3，但没有更新 next index。



1.4. PySyncObj#167-2[①]

这是一个协议级 bug，存在某种情况使得 Leader 中的 match index（图中简称 mi）不单调递增，违反了 Raft 协议要求的安全属性。由于 match index 用于判断哪些 log entry 能 commit，match index 减小可能潜在导致严重的数据丢失或错误。

根本原因：

1. PySyncObj#167-1 中提到的 Leader 收到 AppendEntriesResponse 成功时未设置 next index

[①] "Raft match index is not monotonic · Issue #167 · bakwc/PySyncObj." https://github.com/bakwc/PySyncObj/issues/167 (accessed May 28, 2023).

2. PySyncObj#167-1 中提到的 Leader 发送 AppendEntries 时更新 next index 的优化
3. Leader 收到 AppendEntriesResponse 成功时，未对即将设置 match index 进行大小检查，存在 AppendEntriesResponse 消息中要求设置的 match index 值更小的情况（PySyncObj 使用 TCP 作为传输层，TCP 协议保证了网络消息不会乱序，但仍然存在 match index 更小的情况，如下一个原因中的 bug）

```
if self.__raftState == _RAFT_STATE.LEADER:
    if message['type'] == 'next_node_idx':
        reset = message['reset']
        nextNodeIdx = message['next_node_idx']
        success = message['success']
        currentNodeIdx = nextNodeIdx - 1
        if reset:
            self.__raftNextIndex[node] = nextNodeIdx
        if success:
            # 未对 currentNodeIdx 的大小进行检查
            self.__raftMatchIndex[node] = currentNodeIdx
```

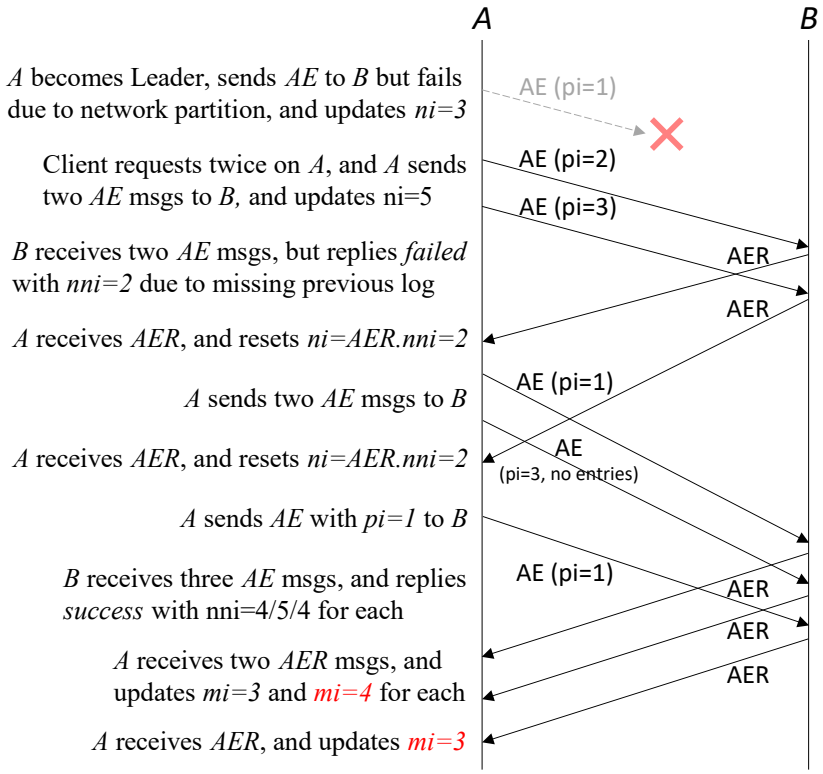
4. Follower 收到包含 log entry 的 AppendEntries 消息时，在返回成功的情况下，错误设置了返回消息中的 nextNodeIdx 为当前最大 log entry 的 index，实际上应该设置为该 index+1

```
if message['type'] == 'append_entries' and message['term']
>= self.__raftCurrentTerm:
    nextNodeIdx = prevLogIdx + 1
    if newEntries:
        nextNodeIdx = newEntries[-1][1] # 少加了 1
    self.__sendNextNodeIdx(node, nextNodeIdx=nextNodeIdx, succ
ess=True)
```

最短触发路径（第 1-6 步与 PySyncObj#167-1 第 1-6 步相同）：

1. A 和 B 初始化为 Follower，log 中仅包含一个 no-op 的 log entry。
2. A 选举成为 Leader，添加一个 no-op 的 log entry 到 log 中，并发送 prev log index 为 1（图中简写为 pi）的 AppendEntries 给 B。发送后，A 对 B 记录的 next index 更新为 3。但由于网络分区，这条 AppendEntries 消息丢失。随后网络恢复。
3. A 接收两条 Client 的 request，分别发送 prev log index 为 2 和 3 的两条 AppendEntries 消息给 B，执行结束后 next index 为 5。
4. B 处理两条 AppendEntries 消息，但由于当前 log 中只有 1 个 log entry，而 prev log index 分别为 2 和 3，因确实日志项返回失败，并在 AppendEntriesResponse 包含 next node index=2。
5. A 收到 AppendEntriesResponse 并将 next index 设置为 2。
6. A 发送 AppendEntries（prev log index 为 1，有 log entries）给 B，并设置 next index 为 5。
7. A 发送 AppendEntries（prev log index 为 3，没有 log entries）给 B。
8. A 收到 AppendEntriesResponse 并将 next index 设置为 2。

9. A 发送 AppendEntries（prev log index 为 1，有 log entries）给 B，并设置 next index 为 5。
10. B 收到三条 AppendEntries 消息，因为 log 能够匹配上，因此全部返回成功。返回的 AppendEntriesResponse 消息中的 next node index 分别是 4/5/4。在 AppendEntries 消息中有 log entries 时（三条消息中的第一条和第三条），由于代码中的 bug，对 next node index 设置有误，少加了 1。
11. A 收到三条 AppendEntriesResponse 消息，因为全部返回了成功，因此更新 match index。更新时，没有对即将设置的 match index 的大小进行检查，处理第二条消息时将 match index 更新为 4，而第三条消息将 match index 更新为 3，出现了 match index 变小的行为。



1.5. PySyncObj#169[①]

这是一个协议级 bug，存在某种情况使得 Leader commit 非当前 term 的 log entry，违反了 Raft 协议的安全属性。Raft 协议论文中举例了这种情况可能带来的严重数据丢失的后果。

根本原因：Leader 提交 log entry 时没有检查该 log entry 的 term 是否等于 Leader 当前的 term。

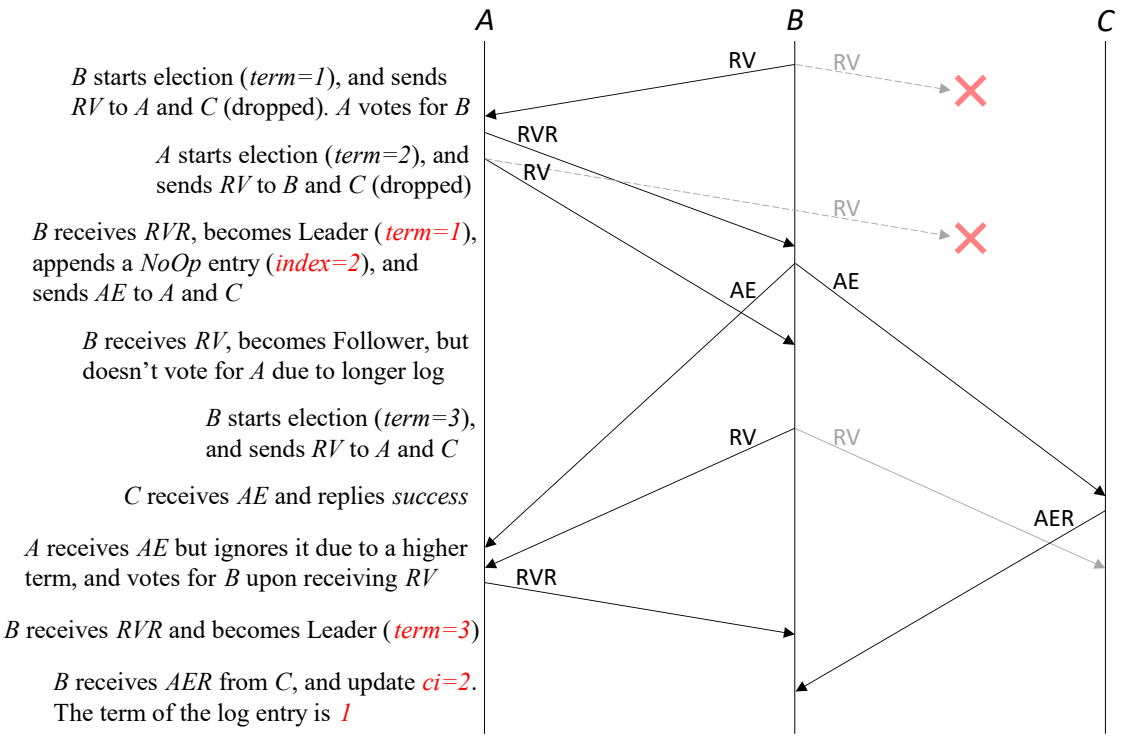
最短触发路径：

1. 初始化 A, B, C 三个节点，C 节点与 A 和 B 存在网络分区。
2. B 发起选举（term 为 1），A 收到 B 的 RequestVote 并对 B 投票。

[①] "Leader commits log entries of older terms · Issue #169 · bakwc/PySyncObj," *GitHub*. https://github.com/bakwc/PySyncObj/issues/169 (accessed May 28, 2023).

3. A 发起选举（term 为 2）。
4. B 收到 A 的投票，成为 Leader，添加一条 NoOp (term=1, index=2) 的 log entry 到 log 中，并发送 AppendEntries 给 A（此时 A 的网络分区已恢复）和 C。
5. B 收到 A 的 RequestVote，由于该选举请求的 term=2，大于当前 term=1，B 变为 Follower。由于 B 的 log 中的 log entry 更新，B 不对 A 投票。
6. B 发起选举（term 为 3）。
7. C 收到来自 B 的 AppendEntries 消息，并回复表示成功的 AppendEntriesResponse。
8. A 收到来自 B 的 AppendEntries 消息，但该消息中的 term=1，比 A 当前的 term=2 小，因此忽略此消息。
9. A 收到 B 的 RequestVote，并对 B 投票。
10. B 收到 A 的投票，成为 Leader (term=3)。
11. B 收到 C 的 AppendEntriesResponse，并更新 commit index 为 2，该 commit index 的 log entry 的 term 为 1，而 B 的当前 term 为 3。

```
while self.__raftCommitIndex < self.__getCurrentLogIndex():
    # 未对 nextCommitIndex 的 log entry 的 term 进行检查
    nextCommitIndex = self.__raftCommitIndex + 1
    count = 1
    for node in self.__otherNodes:
        if self.__raftMatchIndex[node] >= nextCommitIndex:
            count += 1
    if count > (len(self.__otherNodes) + 1) / 2:
        self.__raftCommitIndex = nextCommitIndex
        self.__raftLog.setRaftCommitIndex(self.__raftCommitIndex)
    else:
        break
```



2. RaftOS

RaftOS 是一个基于 Python 语言实现的 Raft 协议库，使用 UDP socket 进行网络消息传输，允许网络包丢失、重复、乱序。

2.1. RaftOS#25[①]

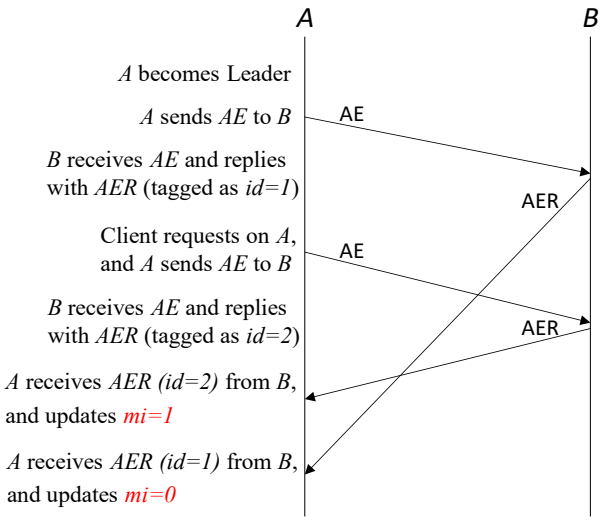
这是一个协议级的 bug，存在某种情况使得 Leader 中的 match index（图中简称 mi）不单调递增，违反了 Raft 协议要求的安全属性。

根本原因：Leader 处理消息 append_entries_response 消息时，没有检查 self.log.match_index 和 data['last_log_index']的大小，直接进行赋值。在网络乱序的情况下，很容易出现消息中的 data['last_log_index']更小的情况

```
    def on_receive_append_entries_response(self, data):
        if not data['success']:
            # ..
        else:
            # 没有进行大小检查就直接赋值
            self.log.next_index[sender_id] = data['last_log_index'] + 1
            self.log.match_index[sender_id] = data['last_log_index']
```

最短触发路径：

1. A 成为 Leader，发送 AppendEntries 给 B
2. B 接收到 AppendEntries，并返回 AppendEntriesResponse，这条消息标记为 id=1
3. A 收到用户请求，添加日志后发送 AppendEntries 给 B
4. B 接收到 AppendEntries，并返回 AppendEntriesResponse，这条消息标记为 id=2
5. A 收到 id=2 的 AppendEntriesResponse，并更新 match index 为 1
6. A 收到 id=1 的 AppendEntriesResponse，并更新 match index 为 0



[①] "Raft match index is not monotonic · Issue #25 · zhebrak/raftos," *GitHub*. https://github.com/zhebrak/raftos/issues/25 (accessed May 28, 2023).

2.2. RaftOS#26[①]

这是一个协议级 bug，存在某种情况使得已提交的 log 没有同步到多数节点上。

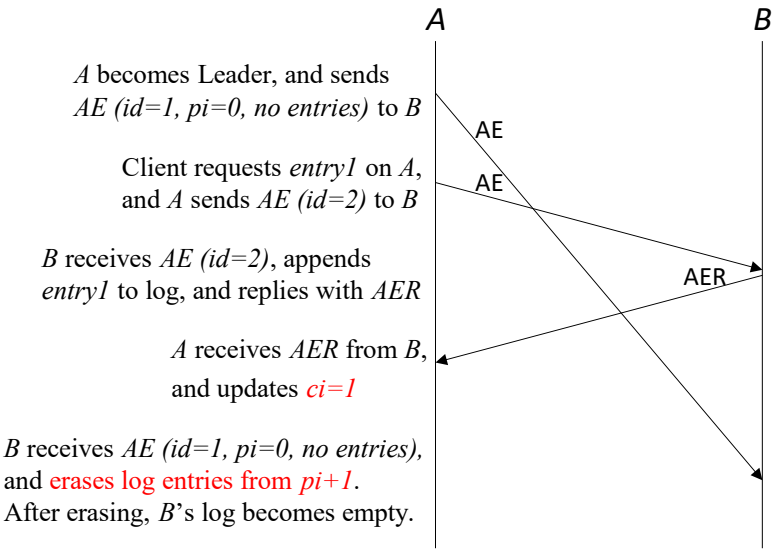
根本原因：Follower 接收 AppendEntries 消息时，认为 self.log.last_log_index!=prev_log_index 是 log 不匹配的情况，并删除 data['prev_log_index']+1 之后的 log。然而，该条件判断与 log 是否匹配无关，log 是否匹配只需要判断 term 是否相等即可，此处的错误判断导致已经匹配的 log 被删除。

```
    def on_receive_append_entries(self, data):
        self.state.set_leader(data['leader_id'])

    # Reply False if log doesn't contain an entry at prev_log_index whos
    e term matches prev_log_term
        try:
            prev_log_index = data['prev_log_index']
            # ...不关心
        except IndexError:
            pass

    # If an existing entry conflicts with a new one (same index but diff
    erent terms),
        # delete the existing entry and all that follow it
        new_index = data['prev_log_index'] + 1
        try:
            # 只需要判断 term，而 self.log.last_log_index != prev_log_index
    不能表明 log 不匹配，这种情况下错误删除了已匹配的 log
            if self.log[new_index]['term'] != data['term'] or (
                self.log.last_log_index != prev_log_index
            ):
                self.log.erase_from(new_index)
        except IndexError:
            pass
```

最短触发路径：

1. A 成为 Leader，并发送 AE (id=1, pi=0) 给 B，这个消息因网络延迟等原因暂未到达 B
2. A 收到 client 的请求，添加 entry1 到 log，并发送 AE (id=2, entry1) 给 B
3. B 收到 AE (id=2)，添加 entry1 到 log 中，并回复 AER
4. A 收到 AER，并更新 commit index 为 1
5. B 收到 AE (id=1, pi=0)，错误认为 log 不匹配，并从 1 开始删除 log entries。删除后，B 的 log 为空。可见在 A 中已 commit 的 log entry 没有被多数节点复制。

[①] "Log may erased incorrectly · Issue #26 · zhebrak/raftos," *GitHub*. https://github.com/zhebrak/raftos/issues/26 (accessed May 28, 2023).



2.3. RaftOS#27[①]

这是一个代码级 bug，存在某种情况，节点尝试从消息中获取一个不存在的 key，导致代码抛出异常。

根本原因：在节点收到 request 消息时（如 ReqeustVote，AppendEntries），如果节点的 term 比消息中的 term 更大，则会直接回复失败，然而回复的消息中没有包含'request_id' key，而接收 AppendEntries response 的方法会检查'request_id'，从而导致代码抛出异常。

```
    def on_receive_function(self, data):
        # ...不关心
        if self.storage.term > data['term'] and
    not data['type'].endswith('_response'):
            # 没有 request_id
            response = {
                'type': '{}_response'.format(data['type']),
                'term': self.storage.term,
                'success': False
            }
```

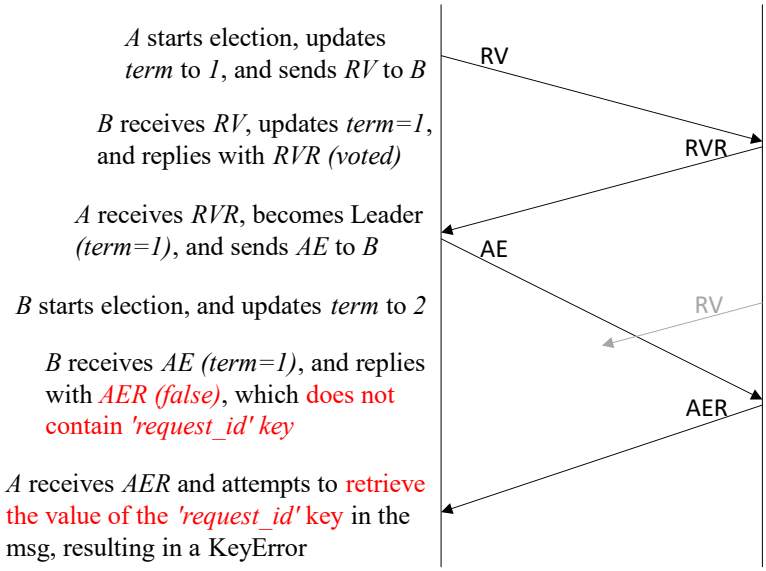
```
    def on_receive_append_entries_response(self, data):
        sender_id = self.state.get_sender_id(data['sender'])
        # 收到 AER 时立刻获取 request_id
        if data['request_id'] in self.response_map:
            self.response_map[data['request_id']].add(sender_id)
```

最短触发路径：

1. A 发起选举，增加 term 为 1，并发送 RequestVote 给 B
2. B 收到 RequestVote，并对 A 投票

① "KeyError in handling append_entries_response message · Issue #27 · zhebrak/raftos," *GitHub*. https://github.com/zhebrak/raftos/issues/27 (accessed May 28, 2023).

3. A 收到 B 的投票，成为 Leader（term=1），并向 B 发送 AppendEntries
4. B 发起选举，增加 term 为 2，并发送 RequestVote 给 A（这条消息不重要）
5. B 收到来自 A 的 AppendEntries，由于消息中的 term=1 比自己的 term=2 小，因此回复 AppendEntriesResponse (false)，然而回复的消息中没有 'request_id' 键值对
6. A 收到 AppendEntriesResponse，试图取出 'request_id' 键值对，然而消息中没有，触发了 KeyError 异常



2.4. RaftOS#30[①]

此 bug 是一个协议级的 bug，存在某种情况，使得 commit index 无法更新，导致集群无法推进工作。

根本原因：由于 Raft 协议的要求，不能 commit 非当前 term 的 index。代码实现中，在更新 commit index 时，使用了一个从前往后遍历的循环，循环在遇到不是当前 term 的 index 时会 break。然而，后面的 index 可能是当前 term 且可以 commit 的，但没有被检查到，导致 commit index 无法推进。

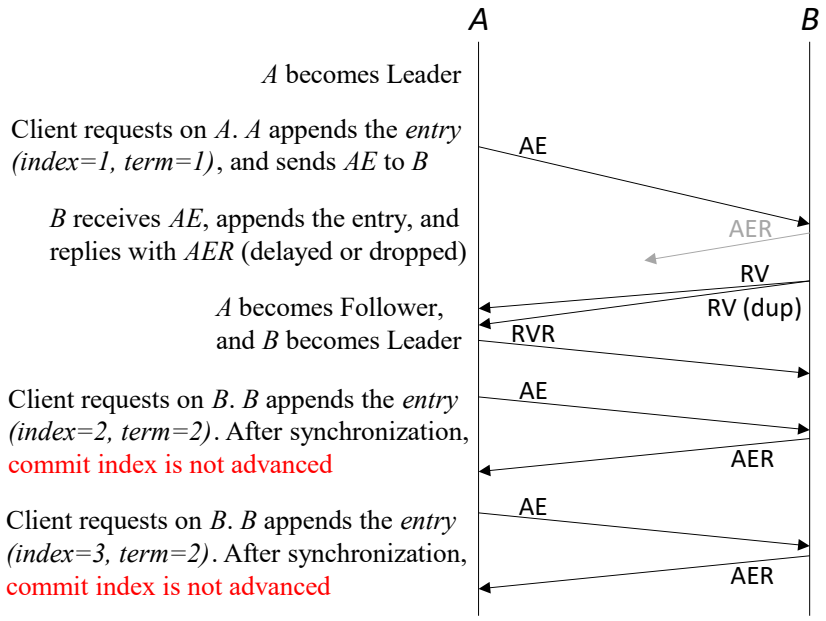
这个 bug 可以被定义为 liveness bug，但也可以通过简单的 safety property 进行检查，我们使用了 safety property 进行检查，得到下面的最短触发路径：

1. A 成为 Leader（term=1）
2. A 收到 client 请求，并添加 index=1, term=1 的 log entry 到 log 中，发送 AppendEntries 给 B
3. B 收到 AppendEntries，并回复 AppendEntriesResponse，然而由于网络延迟或者丢包，该消息暂未被 A 收到
4. B 发起选举，并成为 Leader（term=2）。由于 RaftOS 的特性，A 收到第一个 RequestVote 只变成 Follower，在第二个重复的 RequestVote 时才对 B 投票

① "Change in update commit index by meteam2022 · Pull Request #30 · zhebrak/raftos," *GitHub*. https://github.com/zhebrak/raftos/pull/30 (accessed May 28, 2023).

5. B 收到 client 请求，并添加 index=2, term=2 的 log entry 到 log 中，发送 AppendEntries 给 A，A 收到后回复 AppendEntriesResponse。B 收到 AppendEntriesResponse 后，应该更新 commit index 为 2，然而由于本小节描述的 bug，B 没有更新 commit index
6. 重复第五步，commit index 仍无法更新，发现 liveness bug（也可认为是 safety bug）

```
    def update_commit_index(self):
        commited_on_majority = 0
        for index in range(self.log.commit_index +
    1, self.log.last_log_index + 1):
            commited_count = len([
                1 for follower in self.log.match_index
                if self.log.match_index[follower] >= index
            ])
            is_current_term = self.log[index]['term']
    == self.storage.term
            if self.state.is_majority(commited_count + 1)
    and is_current_term:
                commited_on_majority = index
            else:
                # 当 is_current_term 为 false 是，else 分支触发，break 循环
                # 导致后续可能 commit 的 index 也无法 commit，应改为 continue
                break
        if commited_on_majority > self.log.commit_index:
            self.log.commit_index = commited_on_majority
```



3. WRaft

WRaft 是一个 C 语言实现的 Raft 协议库，该协议库不对底层网络进行假设，因此可注入的错误类型包括 UDP 和 TCP 允许的错误类型。

WRaft 的所有 bug 都提在了一个 PR 中（该 issue 中有 9 个 bug），这是一种不好的风格。对于已经没有积极维护的仓库，增加了开发者理解的难度，更难以得到回复。9 个 bug 中，有

三个违反安全属性可能降低性能但不导致严重 bug（3,5,7，其中 7 是 Redis 开发者提交的 PR 造成的），有两个会导致集群无法推进工作的 liveness bug（8,9，暂未通过 TLA+触发），有一个使用 Valgrind 发现的内存泄漏问题（6，但不确定是否是使用有误，因为 Redis 开发者也提过，但后来 Redis 说自己用错了）。剩下的三个 bug（1,2,4）违反了安全属性，且会导致严重的 bug，在代码中也有明显的错误，在本节中详细描述。

3.1. WRaft#118.1[①]

这是一个协议级的 bug，存在某种情况使得 Follower 收到 AppendEntries 消息后，重复添加已添加的日志项，使得数据异常或损坏。

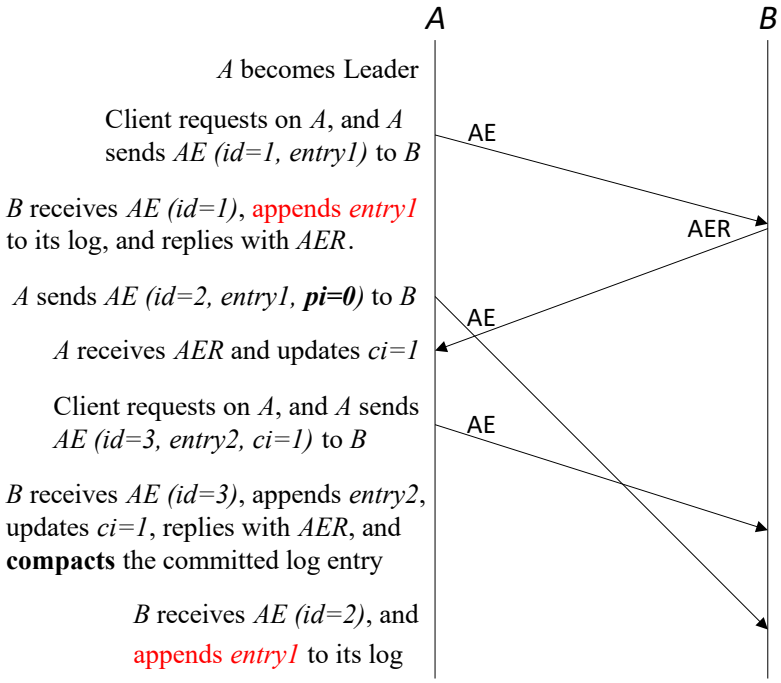
根本原因：当 Follower 对已提交的日志做快照后，如果收到 previous log index（图中简称 pi）为 0 的 AppendEntries 消息时，AppendEntries 消息中的日志项会被无条件添加到 log 中。

最短触发路径（仅展示到触发代码 bug 的位置，由图容易推断模型安全属性必然被违反）：

1. A 成为 Leader
2. A 收到 Client 的请求，添加一个 entry1 到 log，并发送 AppendEntries (entry1, prev_log_idx=0), 消息记为 id=1
3. B 收到 id=1 的 AppendEntries，添加 entry1 到 log，并回复 AppendEntriesResponse 消息
4. A 心跳超时发送 AppendEntries (entry1, **prev_log_idx=0**) 给 B，消息记为 id=2
5. A 收到来自 B 的 AppendEntriesResponse 消息，并更新 commit index 为 1
6. A 收到 Client 的请求，添加一个 entry2 到 log，并发送 AppendEntries (entry2, commit_idx=1)给 B，消息记为 id=3
7. B 收到 id=3 的 AppendEntries，添加 entry2 到 log，更新 commit index 为 1，并回复 AppendEntriesResponse 消息（此消息不关心，图中未画出）
8. B 达到做快照的条件，对已 commit 的 log entry**（即 entry1）从 log 中删除**
9. B 收到 id=2 的 AppendEntries，添加 entry1 到 log
10. 显然，Leader 中只有两个 log entry，而 Follower 中有三个（第一个已 compact），当 Leader 提交第三个 log entry 时，可发现提交的日志在不同节点中不一致（并导致状态机不一致），即违反了模型安全属性

[①] "Fix bugs mainly caused by snapshot by tangruize · Pull Request #118 · willemt/raft," *GitHub*. https://github.com/willemt/raft/pull/118#:~:text=raft_recv_appendentries%23L432 (accessed May 28, 2023).

```
    /* Not the first appendentries we've received */
    /* NOTE: the log starts at 1 */
    // 特殊处理了 previous log index 大于 0 的情况，当等于 0 时，不进入此分支
    if (0 < ae->prev_log_idx)
    {
        raft_entry_t* ety = raft_get_entry_from_idx(me_, ae->prev_log_idx);
        /* Is a snapshot */
        if (ae->prev_log_idx == me->snapshot_last_idx)
            // ...不关心

    /* 2. Reply false if log doesn't contain an entry at prevLogIndex
           whose term matches prevLogTerm (§5.3) */
        else if (!ety)
        {
            // 当 prev log idx 大于 0 时，如果 entry 不存在（未收到或已快照），则返回失败
            __log(me_, node, "AE no log at prev_idx %d", ae->prev_log_idx);
            goto out;
        }
        else if (ety->term != ae->prev_log_term)
            // ...不关心
    }
    r->success = 1;
    r->current_idx = ae->prev_log_idx;
    /* 3. If an existing entry conflicts with a new one (same index
       but different terms), delete the existing entry and all that
       follow it (§5.3) */
    int i;
    for (i = 0; i < ae->n_entries; i++)
    {
        raft_entry_t* ety = &ae->entries[i];
        raft_index_t ety_index = ae->prev_log_idx + 1 + i;
        raft_entry_t* existing_ety = raft_get_entry_from_idx(me_, ety_index);
        if (existing_ety && existing_ety->term != ety->term)
            // ...不关心
        else if (!existing_ety)
            // 如果 entry 不存在，则跳出循环，跳出时的 i 的位置之后的 entries 都会被 append
            break;
        // ...不关心
    }
    /* Pick up remainder in case of mismatch or missing entry */
    for (; i < ae->n_entries; i++)
    {
        // 将上一个循环跳出时 AE 中的 entries 全部添加到 log 中
        e = raft_append_entry(me_, &ae->entries[i]);
        // ...不关心
    }
```



后续：WRaft 的所有 bug 都没有得到开发者的回应（2021 年 8 月 28 日至今）。这个 bug 中对 prev log index 是否大于 0 的判断在 RedisRaft#148[①]中已去除（2022 年 10 月 26 日）。因此可认为 RedisRaft 已独立修复了此 bug（然而在之前的 issue 讨论中，RedisRaft 开发者认为 prev log index 为 0 的情况不会发生。实际上在系统初始运行时会发生，运行过程中不会再出现，可认为触发概率极低，但存在）。DaosRaft 尚未修复此 bug，可尝试验证加入 snapshot 模块后 DaosRaft 是否仍存在此 bug（代价较高），或根据这个 trace，构造 testcase 直接通过 SandTable 让 DaosRaft 执行，看是否能重现（代价较低）。

3.2. WRaft#118.2[②]

这是一个协议级 bug，Follower 收到因 bug 导致的错误 AppendEntries 消息时，更新了 commit index，使得不应该被提交的 log entry 被提交。这些被提交的 log entry 可能在后续同步中被删除，违反了已 commit 的 log 不能被删除的安全属性。另一种更严重的后果是，错误更新了 commit index 的 Follower 后续成为 Leader，使得正确的已 commit 的 log entry 被删除。

根本原因：Leader 发送 AppendEntries 时，对判断发送 AppendEntries 还是 Snapshot 的边界条件判断有误，导致应该发送 Snapshot 的消息发送成了 AppendEntries。然而，由于相关 log 已被 snapshot，AppendEntries 中包含的 entries 为空。Follower 收到该 AppendEntries 时，如果该 AppendEntries 的 previous log index 为 0，则 previous log 匹配上，进入添加 entries 的代码，而 AppendEntries 中的 entries 为空，则不会删除不匹配的 entries，Follower 更新 commit index

① "Refactor raft_recv_appendentries() by tezc · Pull Request #148 · RedisLabs/raft," *GitHub*. https://github.com/RedisLabs/raft/pull/148 (accessed May 28, 2023).

② "Fix bugs mainly caused by snapshot by tangruize · Pull Request #118 · willemt/raft," *GitHub*. https://github.com/willemt/raft/pull/118#:~:text=raft_send_appendentries%23L901 (accessed May 28, 2023).

为 AppendEntries 中的 commit index，该 commit index 的 log entry 可能是 Follower 与 Leader 不匹配的 entry。

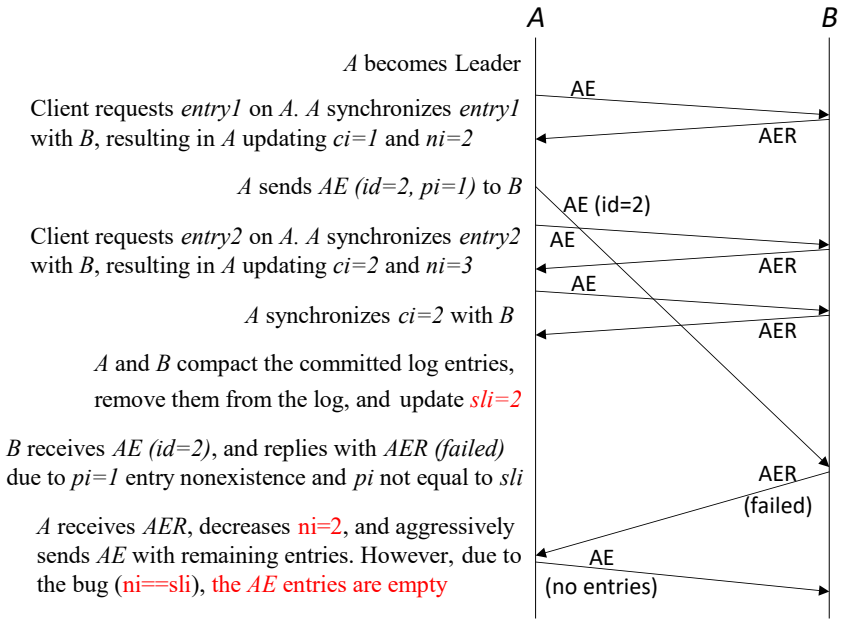
```
int raft_send_appendentries(raft_server_t* me_, raft_node_t* node)
{
    raft_server_private_t* me = (raft_server_private_t*)me_;
    // ...不关心
    raft_index_t next_idx = raft_node_get_next_idx(node);
    /* figure out if the client needs a snapshot sent */
    // 错误判断发送 snapshot 条件，应为 next_idx <= me->snapshot_last_idx
    if (0 < me->snapshot_last_idx && next_idx < me-
>snapshot_last_idx)
    {
        if (me->cb.send_snapshot)
            me->cb.send_snapshot(me_, me->udata, node);
        return RAFT_ERR_NEEDS_SNAPSHOT;
    }
    // ...不关心
}
```

第二个写错的地方，在做快照时，没有对处于边界条件的节点发送快照，但不会造成本小节描述的 bug:

```
int raft_end_snapshot(raft_server_t *me_)
{
    raft_server_private_t* me = (raft_server_private_t*)me_;
    // ...不关心
    for (i = 0; i < me->num_nodes; i++)
    {
        raft_node_t* node = me->nodes[i];
        raft_index_t next_idx = raft_node_get_next_idx(node);
        /* figure out if the client needs a snapshot sent */
        // 相同错误原因，第二个写错的地方，但不会造成严重 bug
        if (0
< me->snapshot_last_idx && next_idx < me->snapshot_last_idx)
        {
            if (me->cb.send_snapshot)
                me->cb.send_snapshot(me_, me->udata, node);
        }
    }
    return 0;
}
```

最短触发路径 1（由于 TLA+的 trace 较复杂，仅展示到触发代码 bug 的位置，由上述描述可知，此 bug 会导致本小节描述的安全属性违反）：

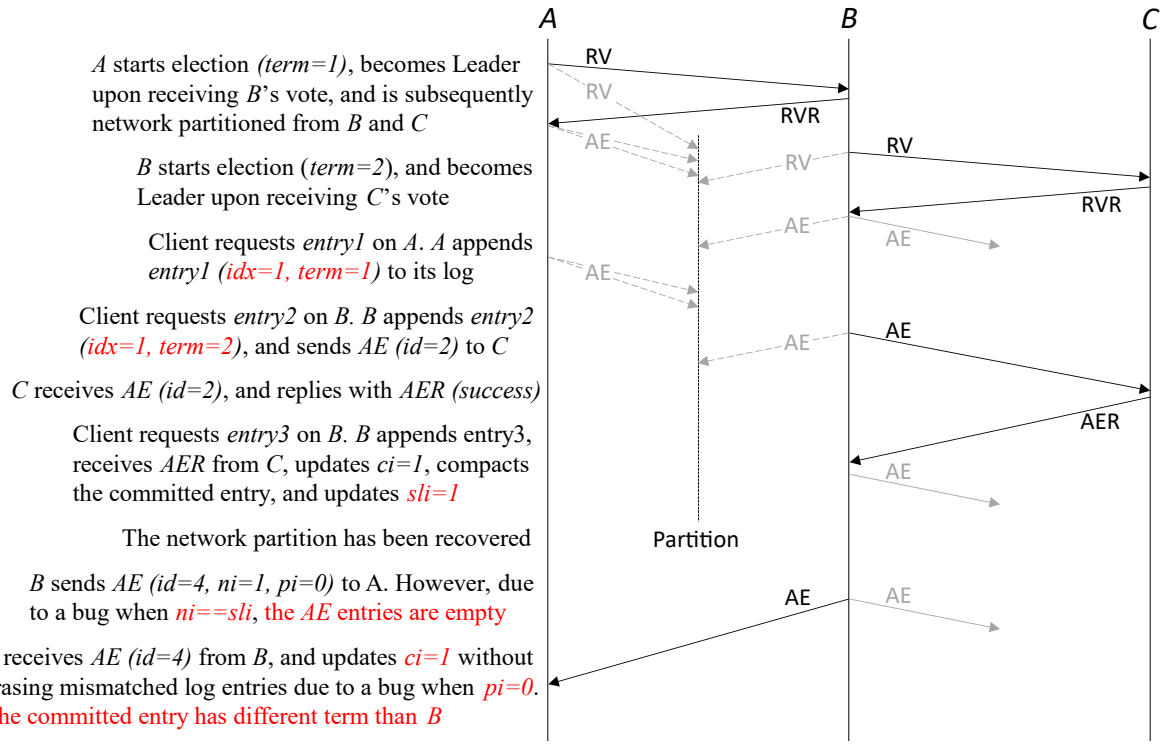
1. A 成为 Leader
2. A 收到 client 的请求，将 entry1 添加到 log，并向 B 同步日志，同步完成后，A 更新 commit index 为 1，A 更新 A 对 B 的 next index 为 2
3. A 心跳超时发送 AppendEntries 给 B，该消息的 previous log index 为 1，记为 AE (id=2)，由于网络延迟，暂时未到达 B
4. A 收到 client 的请求，将 entry2 添加到 log，并向 B 同步日志，同步完成后，A 更新 commit index 为 2，A 更新 A 对 B 的 next index 为 3
5. A 心跳超时发送 AppendEntries 给 B，该消息的 commit index 为 2，同步完成后，B 更新 commit index 为 2
6. A 和 B 达到做快照的条件，更新 snapshot last index 为 2，分别将已提交的 log entries 从 log 中删除（即删除了所有的日志项）
7. B 收到 AE (id=2)，由于 previous log index 为 1 位置的 log entry 不存在，且 previous log index 不等于 snapshot last index，B 回复 AppendEntriesResponse (failed)
8. A 收到 AppendEntriesResponse (failed)，递减 next index 为 2。由于 next index 不大于最大日志长度，A 认为 B 的日志并未达到同步状态，进行快速同步。由于本小节描述的 bug，即 next index == snapshot last index 的情况，A 向 B 发送 AppendEntries 而不是 InstallSnapshot，发送过程中，由于 next index 所在位置的 log entry 已被删除，该 AppendEntries 中包含了空的 entries，与快速同步的目的不符。
9. 显然，如果 B 中有与 A 不相同但未 commit 的日志项（即 index 相同，term 不同的日志项），如果错误发送的 AppendEntries 消息中的 commit index 大于等于该不相同日志项的 index，且在某种情况下 log 被判断为匹配（例如，previous log index 为 0 情况下 log 无条件匹配），由于消息中的 entries 为空，不会使得不相同的日志项被删除，则会导致该日志项被错误的提交，导致更严重的后果。



最短触发路径 2（展示到已提交的日志项在不同节点有不同的 term，结合了 WRaft#118.1 和 WRaft#118.2 两个 bug）：

1. A 发起选举，收到 B 的投票后成为 Leader（term=1）
2. A 与 B 和 C 网络分区
3. B 发起选举，收到 C 的投票后成为 Leader（term=2）
4. A 收到 client 的请求，并将 entry1(index=1, term=1) 的日志项添加到 log，由于网络分区，A 发送的 AppendEntries 无法到达
5. B 收到 client 的请求，将 entry2(index=1, term=2) 的日志项添加到 log，并向 C 发送 AppendEntries (id=2)
6. 由于网络包乱序到达，C 收到 id 为 2 的 AppendEntries（减少回复先前的 AE 的一步操作），并回复 AppendEntriesResponse 成功
7. B 收到 client 的请求，将 entry3 的日志项添加到 log（WRaft 要求至少两个日志项才能做快照）
8. B 收到来自 C 的 AppendEntriesResponse，更新 commit index 为 1
9. B 做快照，更新 snapshot last index 为 1，snapshot last term 为 2，并将提交的日志项（即 index 为 1 的日志项）从 log 中删除
10. A 与 B 和 C 的网络分区恢复
11. B 心跳超时，此时 B 对 A 的 next index 为 1，由于本小节中描述的 bug，即 next index 等于 snapshot last index 时，B 错误发送 AppendEntries 给 A（应该发送 InstallSnapshot）。该 AppendEntries 消息中，commit index 为 1，previous log index 为 0，entries 因为已被压缩无法获取到所以为空
12. A 收到来自 B 的 AppendEntries，由于 WRaft#118.1 中的 bug，即 previous log index 为 0 时，A 认为 previous log 匹配。而消息中的 entries 为空，导致不匹配日志项（即 index 为 1 的日志项）未被删除。A 更新 commit index 为消息中的 commit index，即为 1。此时，A 和 B 的 index 为 1 不匹配日志项均在两个节点提交，可造成严重的数据损坏
13. 在 liveness 方面，当 A 的 log 为空时（在节点初始化时，或刚加入集群时容易出现），A 收到错误的 AppendEntries，回复的 AppendEntriesResponse 使得 B 更新 next index 后仍然不变，下次继续发送错误的 AppendEntries，如果 snapshot last index 不改变，将导致 A 和 B 的状态无法同步。在 WRaft#118.8[①]的作用下（发送心跳过程中，如果循环中出现发送 InstallSnapshot，则会取消后面的节点的 AppendEntries 的发送），集群的容错能力可能下降。

[①] "Fix bugs mainly caused by snapshot by tangruize · Pull Request #118 · willemt/raft," *GitHub*. https://github.com/willemt/raft/pull/118#:~:text=raft_send_appendentries_all%23L952 (accessed May 28, 2023).



后续：RedisRaft 在 2021 年 8 月 26 日提交了关于这个 bug 的 PR（RedisRaft#47[①]），可认为与我们同时独立修复（我们 2021 年 8 月 28 日提的 PR，但在 RedisRaft 提 PR 前已发现）。DaosRaft 在 2018 年 7 月 12 日提交了一个 PR 包含了对这个 bug 的修复（DaosRaft#10[②]）。

关于 WRaft#118.8[③]的后续：在 RedisRaft#49[④]中得到开发者的确认，在 RedisRaft#68[⑤]（2021 年 11 月 15 日）已修复。在 DaosRaft#10 中已修复。

修复过程中的发现：WRaft 中有一个测试用例的函数名叫做 **TestRaft_leader_sends_appendentries_when_node_next_index_was_compacted**，意思是在 next index 被 compact 的情况下，应该发送 AppendEntries，**表明开发者经过仔细思考，认为这种边界情况应该按照这种方式处理**。然而，在缺少 oracle 的情况下，人的想法可能是错误的，导致写出的测试用例是错误的。RedisRaft 修复时将该测试用例函数名改为 TestRaft_leader_sends_snapshot_when_node_next_index_was_compacted，DaosRaft 修复时虽然没有更改此测试用例的函数名，但测试用例的内容改成了发送 snapshot。

---

① "Send snapshot when node's next index is snapshot's last included index by tezc · Pull Request #47 · RedisLabs/raft," *GitHub*. https://github.com/RedisLabs/raft/pull/47 (accessed May 28, 2023).

② "Log compaction by liw · Pull Request #10 · daos-stack/raft," *GitHub*. https://github.com/daos-stack/raft/pull/10 (accessed May 28, 2023).

③ "Fix bugs mainly caused by snapshot by tangruize · Pull Request #118 · willemt/raft," *GitHub*. https://github.com/willemt/raft/pull/118#:~:text=raft_send_appendentries_all%23L952 (accessed May 28, 2023).

④ "Found bugs mainly caused by snapshot · Issue #49 · RedisLabs/raft," *GitHub*. https://github.com/RedisLabs/raft/issues/49 (accessed May 28, 2023).

⑤ "Snapshot RPC by tezc · Pull Request #68 · RedisLabs/raft," *GitHub*. https://github.com/RedisLabs/raft/pull/68 (accessed May 28, 2023).

3.3. WRaft#118.4[①]

这是一个协议级 bug，导致 current term 不单调。

根本原因：载入 snapshot 时，未对节点 current term 和 snapshot 中的 current term 进行比较，直接赋值为 snapshot 中的 current term，如果 snapshot 中的较小，则会导致 current term 不单调。

```
int raft_begin_load_snapshot(
    raft_server_t *me_,
    raft_term_t last_included_term,
    raft_index_t last_included_index)
{
    raft_server_private_t* me = (raft_server_private_t*)me_;
    // ...不关心
    // 直接对 current_term 进行赋值，但没检查所赋的值是否更大
    me->current_term = last_included_term;
}
```

最短触发路径：

1. A 与 B 和 C 网络分区
2. B 成为 Leader（term=1）
3. B 收到两次 client 的请求，并向 C 同步，同步完成后，B 和 C 的 commit index 为 2
4. A 与 B 和 C 的网络分区恢复
5. C 发起选举，得到 A 的投票成为 Leader（term=2），并向 A 发送 AppendEntries。A 更新了 term=2
6. C 做快照，并将两条已提交的日志项从 log 中删除，并更新 snapshot last term 为 1
7. A 收到 AppendEntries，由于日志为空，匹配失败，回复 AppendEntriesResponse (failed)
8. C 收到 AppendEntriesResponse，发送 Snapshot (snapshot last term=1)给 A
9. A 收到 Snapshot，更新 term 为 snapshot last term，即为 1

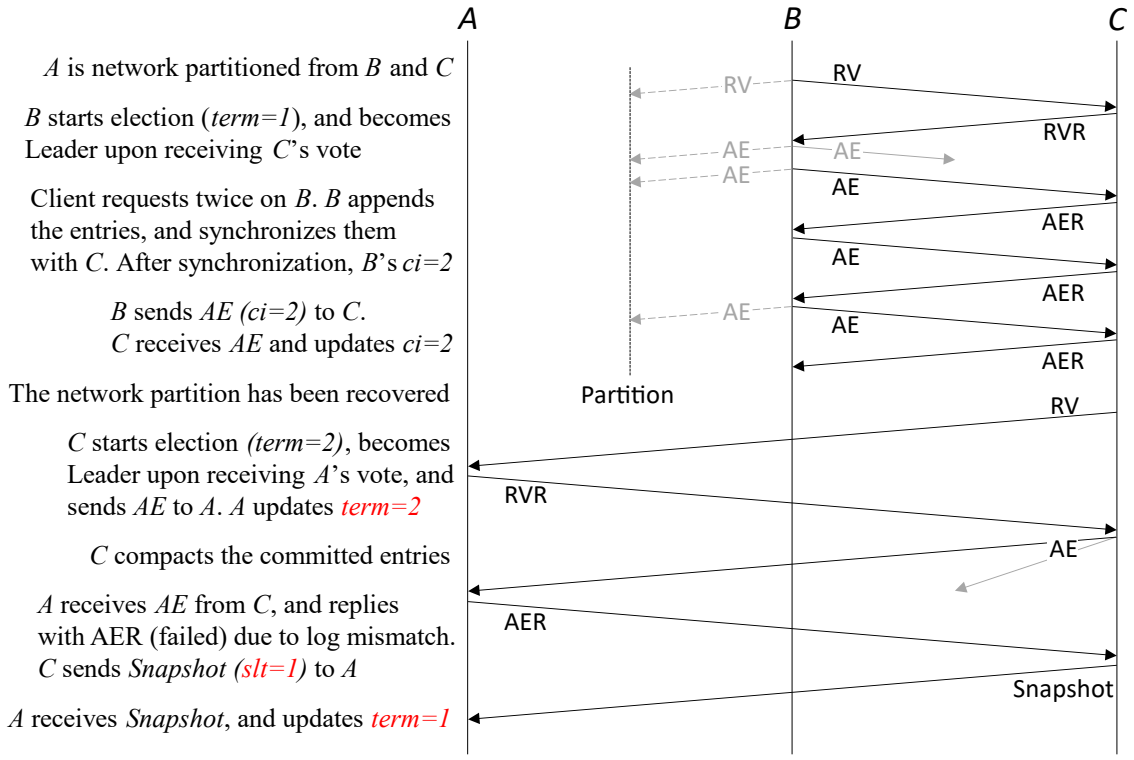
后续：DaosRaft#10[②]（2018 年 7 月 12 日）中重写了大部分与 snapshot 相关的逻辑，包括删除了对 current_term 的赋值，RedisRaft@d23f390[③]（2020 年 5 月 26 日）中修复了此 bug。

---

① "Fix bugs mainly caused by snapshot by tangruize · Pull Request #118 · willemt/raft," *GitHub*. https://github.com/willemt/raft/pull/118#:~:text=raft_begin_load_snapshot%23L1383 (accessed May 28, 2023).

② "Log compaction by liw · Pull Request #10 · daos-stack/raft," *GitHub*. https://github.com/daos-stack/raft/pull/10 (accessed May 28, 2023).

③ "Fix: safety issues loading a snapshot. · RedisLabs/raft@d23f390," *GitHub*. https://github.com/RedisLabs/raft/commit/d23f3903a7e5394757cdd011c50c856dc5eb9cf1 (accessed May 29, 2023).



4. DaosRaft

DaosRaft 是 WRaft 的衍生版本，用于 Intel 的 Daos 存储系统。该协议库不对底层网络进行假设，因此可注入的错误类型包括 UDP 和 TCP 允许的错误类型。

4.1. DaosRaft#70[①]

这是一个代码级 bug，存在某种特定配置的情况下，选主阶段触发 assertion，导致系统崩溃。

根本原因：代码块 1 中，如果 me->timeout_elapsed < me->election_timeout 条件不成立，则代码不会进入 if 分支。如果配置 Leader 发送心跳的超时大于等于选举超时，则可使得上述条件不成立。

代码块 1：

```
    /* Reject request if we have a leader */
    // 当 timeout_elapsed >= election_timeout 时，不进入此分支
    if (me->leader_id != -1 && me->leader_id != vr->candidate_id &&
        me->timeout_elapsed < me->election_timeout)
    {
        r->vote_granted = 0;
        goto done;
    }
```

[①] "Reject request vote if self is leader by liw · Pull Request #70 · daos-stack/raft," *GitHub*. https://github.com/daos-stack/raft/pull/70 (accessed May 29, 2023).

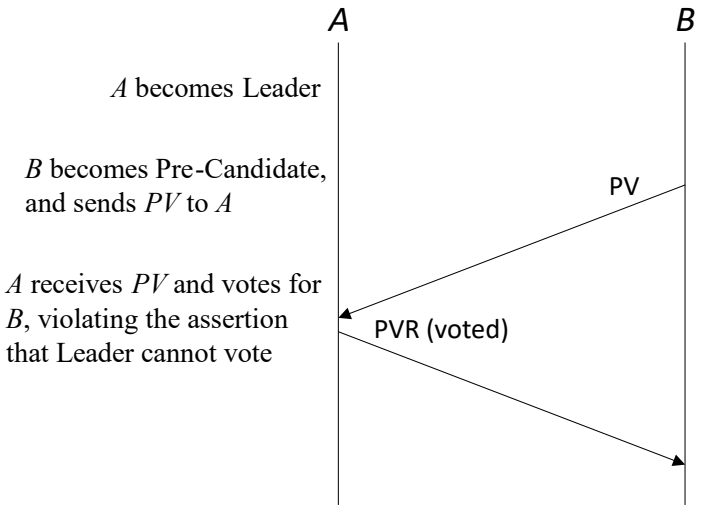
在 Leader 收到 PreVote 消息时，代码块 2 要求 Leader 不能给其他节点投票，然而当代码块 1 中的情况出现时，Leader 没有 goto 到 done 的位置，因此进入了投票的代码，触发了 assertion。

代码块 2:

```
    if (__should_grant_vote(me, vr))
    {

    /* It shouldn't be possible for a leader or prevoted candidate to gr
    ant a vote
         * Both states would have voted for themselves
         * A candidate may grant a prevote though */
        assert(!raft_is_leader(me_) && (!raft_is_candidate(me_) || me-
    >prevote || vr->prevote));
    }
```

最短触发路径:

1. 配置 Leader 的发送心跳的超时等于最小选举超时
2. A 成为 Leader
3. B 成为 Pre-Candidate，并发送 PreVote 给 A
4. A 收到 PreVote，此时 A 记录的 timeout_elapsed 等于 election_timeout。A 对 B 的 PreVote 投票，触发了 Assertion



5. RedisRaft

RedisRaft 是 WRaft 的衍生版本，是 Redis 的一个可插拔模块。该协议库不对底层网络进行假设，因此可注入的错误类型包括 UDP 和 TCP 允许的错误类型。

目前暂未发现 RedisRaft 的 bug.

6. Xraft

Xraft 是一个基于 Java 语言实现的 Raft 协议库，使用的 RPC 的底层网络基于 TCP 协议，因此只能注入网络分区错误。

6.1. Xraft#33[①]

这是一个协议级 bug，会导致出现多个相同 term 的 Leader，是 Raft 协议中不允许出现的一种最严重 bug。

根本原因：Candidate 在接收到 ReqeustVoteResponse 时，没有对消息中的 term 的大小进行检查。如果 term 较小，Candidate 仍会接受 ReqeustVoteResponse，并成为 Leader。

```
    private void doProcessRequestVoteResult(RequestVoteResult result) {
        // step down if result's term is larger than current term
        if (result.getTerm() > role.getTerm()) {
            becomeFollower(result.getTerm(), null, null, true);
            return;
        }
        // check role
        if (role.getName() != RoleName.CANDIDATE) {
            logger.debug("receive request vote result and current role i
    s not candidate, ignore");
            return;
        }
        // do nothing if not vote granted
        if (!result.isVoteGranted()) {
            return;
        }
        // 以上检查中没有对 term 的检查，下面进入统计投票代码
        int currentVotesCount =
    ((CandidateNodeRole) role).getVotesCount() + 1;
        int countOfMajor = context.group().getCountOfMajor();
        logger.debug("votes count {}, major node count {}", currentVotes
    Count, countOfMajor);
        // ...
    }
```

最短触发路径：

1. A 与 B 和 C 的网络连接有较大的延迟，图中虚线表示因延迟而尚未达到的网络消息
2. A 选举超时（term 为 1）发送 RequestVote 给 B 和 C
3. B 接收到来自 A 的 RequestVote，更新 term 为 1，并回复 RequestVoteResponse 对 A 投票
4. A 再次选举超时（term 为 2）
5. A 收到来自 B 的 RequestVoteResponse（term 为 1），并成为 Leader（term 为 2）

[①] "Missing check for `result.getTerm() == role.getTerm()` in `doProcessRequestVoteResult()` can result in two leaders with the same term. · Issue #33 · xnnyygn/xraft," *GitHub*. https://github.com/xnnyygn/xraft/issues/33 (accessed May 29, 2023).

6. B 选举超时（term 为 2），发送 RequestVote 给 A 和 C
7. C 接收到 B 的 RequestVote，并回复 RequestVoteResponse 对 B 投票
8. B 收到来自 C 的 RequestVoteResponse（term 为 2），并成为 Leader（term 为 2）