BusyReplying Bug

- In VENTOS, during the execution of operations, the protocol only considers the messages about the on-going operation.
- Since this logic makes the delivery of unnecessary messages, it needs to be resolved by adding a common message handler.
- Ex) during $Split$ operation, only the messages of $Split$ micro-operations are considered, and other messages, such as $MERGE\_REQUEST$ are ignored.

```
else if(vehicleState == state_waitForSplitReply)
{
    if(wsm->getUCommandType() == SPLIT_ACCEPT &&
```

```
else if(vehicleState == state_waitForAck)
{
    if (wsm->getUCommandType() == ACK &&
    {
```

```
else if(vehicleState == state_waitForCHANGEPL)
{
    if (wsm->getUCommandType() == CHANGE_PL &&
    {
```

```
else if(vehicleState == state_waitForSplitDone)
{
    if(wsm->getUCommandType() == SPLIT_DONE &&
    {
```

```
else if(vehicleState == state_platoonFollower)
{
    // splitting vehicle receives a SPLIT_REQ f
    if (wsm->getUCommandType() == SPLIT_REQ &&
    {
```

MERGE_REQUEST Attempt bug

- When a platoon leader requests $Merge$ and the receiver rejects or ignores $MERGE_REQUEST$s more than three times, the sender should no longer send messages.
- However, in VENTOS, since they just change the mergeReqAttempts variable to 0, the sender requests Merge again to the same vehicle by the end of the simulation.
- This bug can be solved by storing the ids of vehicles exceeding three times of the $MERGE_REQUEST$s.

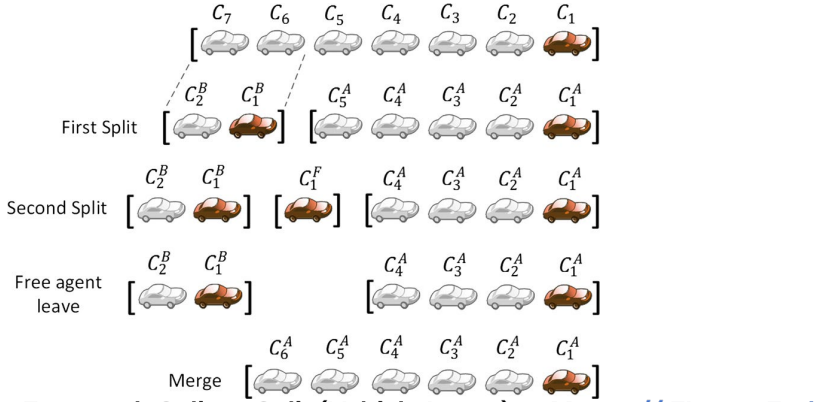
```
if(msg == plnTIMER1) // plnTIMER1: waitForMergeRequest
{
    if(vehicleState == state_waitForMergeReply)
    {
        // leader does not response after three re-attempts!
        if(mergeReqAttempts >= 3)
        {
            mergeReqAttempts = 0;

            setVehicleState(state_platoonLeader);
        }
        else
        {
            setVehicleState(state_sendMergeReq);

            merge_BeaconFSM();
        }
    }
}
```

FollowerLeaveProtocol bug

- In the paper on VENTOS protocol, they described that the $Middle\ Follower\ Leave$ consists of the sequential operation of $Split$, $Split$, and $Merge$.
- However, the last $Merge$ operation is executed by $OptSize$ policy and it causes that $Merge$ is requested before the end of preceding operations, and thus triggers several unnecessary messages.
- Therefore, it must be implemented that the $Merge$ is executed in the proper order according to the specified protocol.



Expected: Split -> Split (Vehicle Leave) -> Merge // FLeave End

Real: Split -> Split (Vehicle Leave) // FLeave End -> Merge

```
// middle follower wants to leave (we need two splits)
else
{
    RemainingSplits = 2;

    // start the first split
    splittingDepth = wsm->getValue().myPltDepth + 1;
    splittingVehicle = plnMembersList[splittingDepth];
    splitCaller = 1;  // Notifying split that follower leave is the caller

    setVehicleState(state_sendSplitReq);

    split_DataFSM();
}
```

LeavedVehList bug

- During or after $Leave$ operation in VENTOS, we found that a new leader or rear leader sent messages to leaved leader vehicle.
- Since, in VENTOS protocol, when a vehicle leaves a platoon, the vehicle is regarded as an unconnected vehicle, it can not reply to any message.
- Therefore, it is necessary to prevent a vehicle from sending several redundant messages, such as $MERGE_REQUEST$s by $OptSize$ policy to leaved vehicles.
- We can solve this bug by storing the ids of vehicles that request $Leave$ in $MERGE\_REQUEST$ attempt code.

LeaveSplitCaller bug (1/2)

- For dealing with various $Leave$ operations in VENTOS, their protocol uses splitCaller variable to check the $Leave$ cases.
  - Ex) splitCaller – 1: $Follower\ Leave$, 0: $Leader\ Leave$, -1: $Split$
- However, in the second $Split$ call for $Middle\ Follower\ Leave$, the splitCaller needs to be -1 separate from the $End\ Follower\ Leave$ call.
- For the $Merge$ operation to be performed by $Middle\ Follower\ Leave$, the leader must accept $MERGE\_REQUEST$ in a non-busy state after the second split ends.
- But now, since the second $Split$ is called with splitCaller=1, the leader keeps busy even after the $Split$ ends.

```
else if(RemainingSplits == 1)
{
    // start the second split
    splittingDepth = plnSize - 1;
    splittingVehicle = plnMembersList[splittingDepth];
    splitCaller = 1;  // Notifying split that follower leave is the caller

    setVehicleState(state_sendSplitReq);

    split_DataFSM();
}
```

LeaveSplitCaller bug (2/2)

- In current implementation, since the second $Split$ is called with splitCaller=1, the leader keeps busy even after the $Split$ ends.
- To solve this bug, the protocol needs to distinguish the second $Split$ call of $Middle\ Follower\ Leave$ and the first $Split$ call of $End\ Follower\ Leave$.

```
else if(RemainingSplits == 1)
{
    // start the second split
    splittingDepth = plnSize - 1;
    splittingVehicle = plnMembersList[splittingDepth];
    splitCaller = 1;  // Notifying split that follower leave is the caller

    setVehicleState(state_sendSplitReq);

    split_DataFSM();
}
```

ChangeVehStateLastly bug

- In $Split$ operation in VENTOS, the last micro-operation to notify that the $Split$ is finished is $GAP\_CREATED$ or $SPLIT\_END$.
- However, in the VENTOS protocol, the state of the split vehicle is changed in $CHANGE\_PL$.
- The problem is that the busy state of the vehicle is not changed yet.
- Due to this bug, we found quite a few examples of $MERGE_REQUEST$s in the execution of other operations.
- We can solve this bug by synchronizing the state and busy change.

```
if (wsm->getUCommandType() == CHANGE_PL && wsm->
{
    cancelEvent(plnTIMER5);

    // save my old platoon leader id for future
    oldPlnID = myPlnID;

    // I am a free agent now!
    myPlnID = wsm->getValue().newPltLeader;
    myPlnDepth += wsm->getValue().newPltDepth;
    plnSize = 1;
```