Canova Tech

*The Art of Silicon Sculpting*

# PLCA Corner-case Fixes

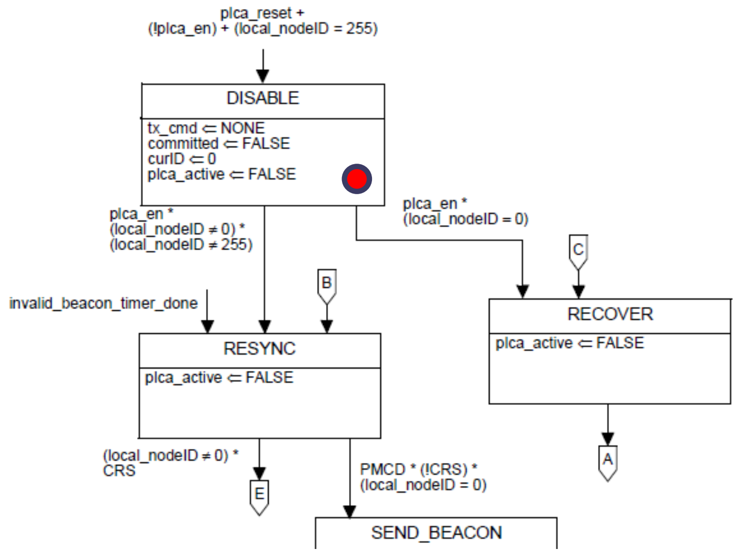
**Feb, 23rd 2022**

## Outline

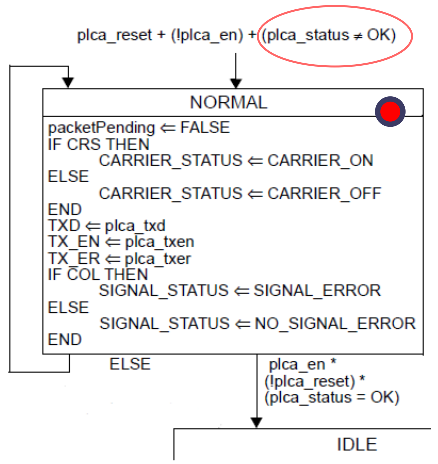
- Since 802.3cg was approved, a number of rare (unwanted) corner-case behaviors were found in the PLCA state diagrams
  - These are really unwanted behaviors that are not covered by the PLCA state diagrams model defined in Clause 148
  - Reasonable implementations already worked around these flaws/limitations
  - Still, I think the model should be updated to cover all cases and be consistent
- This presentation shows what these corner cases are and suggests fixes
  - In no case these changes break interoperability with what is currently defined in Clause 148

## #1: Data corruption when dynamically switching from plain CSMA/CD to PLCA

## #1: problem


plca_reset + (!plca_en) + (local_nodeID = 255)
DISABLE
tx_cmd ⇐ NONE
committed ⇐ FALSE
curID ⇐ 0
plca_active ⇐ FALSE
plca_en * (local_nodeID ≠ 0) * (local_nodeID ≠ 255)
plca_en * (local_nodeID = 0)
C
B
invalid_beacon_timer_done
RECOVER
plca_active ⇐ FALSE
RESYNC
plca_active ⇐ FALSE
(local_nodeID ≠ 0) * CRS
E
PMCD * (!CRS) * (local_nodeID = 0)
A
SEND_BEACON

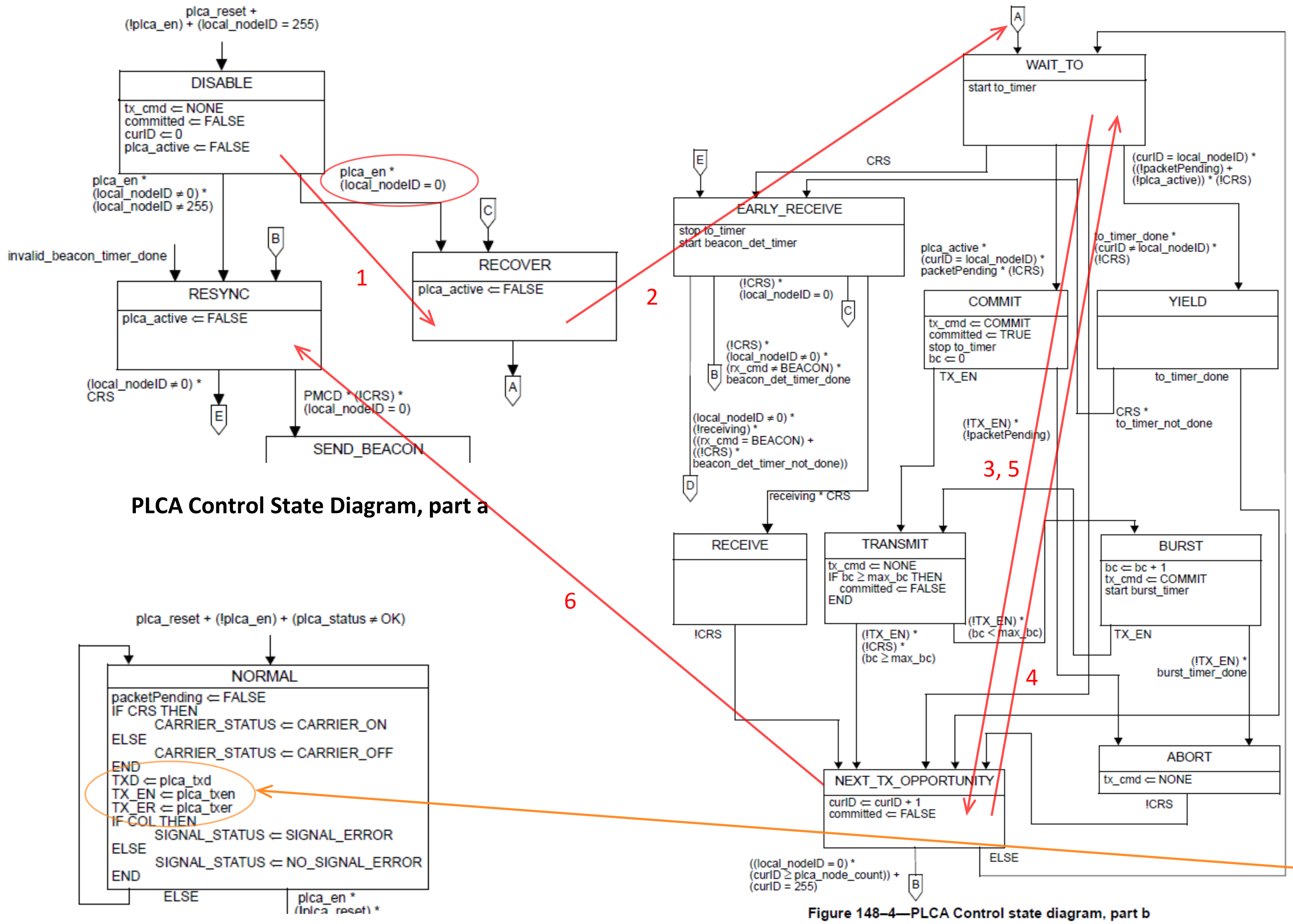
**PLCA Control State Diagram, part a**


plca_reset + (!plca_en) + (plca_status ≠ OK)
NORMAL
packetPending ⇐ FALSE
IF CRS THEN
CARRIER_STATUS ⇐ CARRIER_ON
ELSE
CARRIER_STATUS ⇐ CARRIER_OFF
END
TXD ⇐ plca_txd
TX_EN ⇐ plca_txen
TX_ER ⇐ plca_txer
IF COL THEN
SIGNAL_STATUS ⇐ SIGNAL_ERROR
ELSE
SIGNAL_STATUS ⇐ NO_SIGNAL_ERROR
END
ELSE
plca_en * (!plca_reset) * (plca_status = OK)
IDLE

**PLCA DATA State Diagram**

- Assuming the coordinator node (local_nodeID = 0) has PLCA enabled (plca_en = TRUE) but no BEACON has been sent so far (plca_status = FAIL)
  – This may happen when enabling PLCA on-the-fly
- Because of plca_status being FAIL, the PLCA DATA State Diagram is stuck in NORMAL state
  – That is, “plain” CSMA/CD behavior

# #1: problem



plca_reset + (!plca_en) + (local_nodeID = 255)
DISABLE
tx_cmd ⇐ NONE
committed ⇐ FALSE
curID ⇐ 0
plca_active ⇐ FALSE
plca_en * (local_nodeID ≠ 0) * (local_nodeID ≠ 255)
plca_en * (local_nodeID = 0)
invalid_beacon_timer_done
RESYNC
plca_active ⇐ FALSE
RECOVER
plca_active ⇐ FALSE
(local_nodeID ≠ 0) * CRS
PMCD * (!CRS) * (local_nodeID = 0)
SEND_BEACON
A
B
C
D
E
1
2
3, 5
4
6
WAIT_TO
start to_timer
CRS
EARLY_RECEIVE
stop to_timer
start beacon_det_timer
(!CRS) * (local_nodeID = 0)
(!CRS) * (local_nodeID ≠ 0) * (rx_cmd ≠ BEACON) * beacon_det_timer_done
(local_nodeID ≠ 0) * (!receiving) * ((rx_cmd = BEACON) + ((!CRS) * beacon_det_timer_not_done))
receiving * CRS
plca_active * (curID = local_nodeID) * packetPending * (!CRS)
(curID = local_nodeID) * ((!packetPending) + (!plca_active)) * (!CRS)
to_timer_done * (curID ≠ local_nodeID) * (!CRS)
COMMIT
tx_cmd ⇐ COMMIT
committed ⇐ TRUE
stop to_timer
bc ⇐ 0
TX_EN
YIELD
to_timer_done
CRS * to_timer_not_done
(!TX_EN) * (!packetPending)
RECEIVE
!CRS
TRANSMIT
tx_cmd ⇐ NONE
IF bc ≥ max_bc THEN
committed ⇐ FALSE
END
(!TX_EN) * (bc < max_bc)
(!TX_EN) * (!CRS) * (bc ≥ max_bc)
BURST
bc ⇐ bc + 1
tx_cmd ⇐ COMMIT
start burst_timer
TX_EN
(!TX_EN) * burst_timer_done
NEXT_TX_OPPORTUNITY
curID ⇐ curID + 1
committed ⇐ FALSE
ABORT
tx_cmd ⇐ NONE
!CRS
((local_nodeID = 0) * (curID ≥ plca_node_count)) + (curID = 255)
ELSE
plca_reset + (!plca_en) + (plca_status ≠ OK)
NORMAL
packetPending ⇐ FALSE
IF CRS THEN
CARRIER_STATUS ⇐ CARRIER_ON
ELSE
CARRIER_STATUS ⇐ CARRIER_OFF
END
TXD ⇐ plca_txd
TX_EN ⇐ plca_txen
TX_ER ⇐ plca_txer
IF COL THEN
SIGNAL_STATUS ⇐ SIGNAL_ERROR
ELSE
SIGNAL_STATUS ⇐ NO_SIGNAL_ERROR
END
ELSE
plca_en * (!plca_reset) *

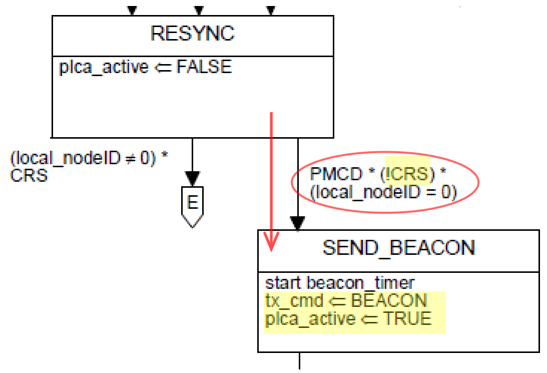
**PLCA Control State Diagram, part a**

Figure 148–4—PLCA Control state diagram, part b

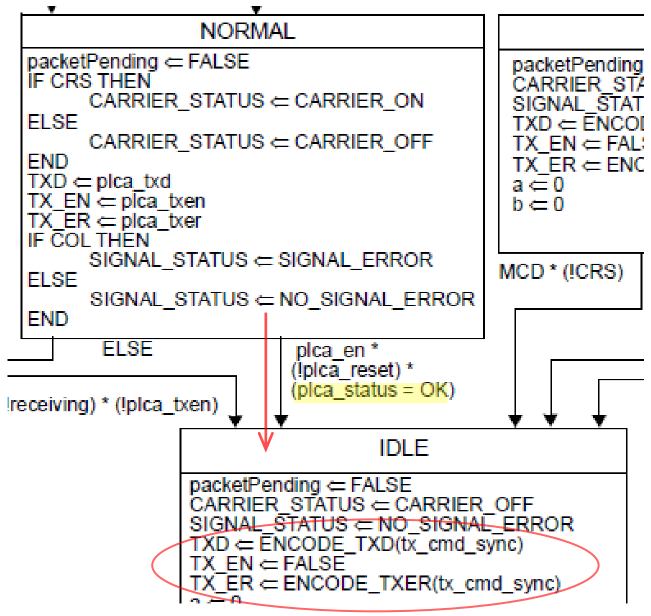
**PLCA DATA State Diagram**

- The PLCA Control State Diagram moves through RECOVER and WAIT_TO states (1, 2) then waits for one cycle of TOs (repeat 3, 4) before entering RESYNC state (5, 6)
- Note that during this time, plca_active = FALSE, hence plca_status = FAIL
- If the MAC initiates a frame transmission at this time, it would be sent immediately (provided that CRS allows it)

## #1: problem



PLCA Control State Diagram, part a

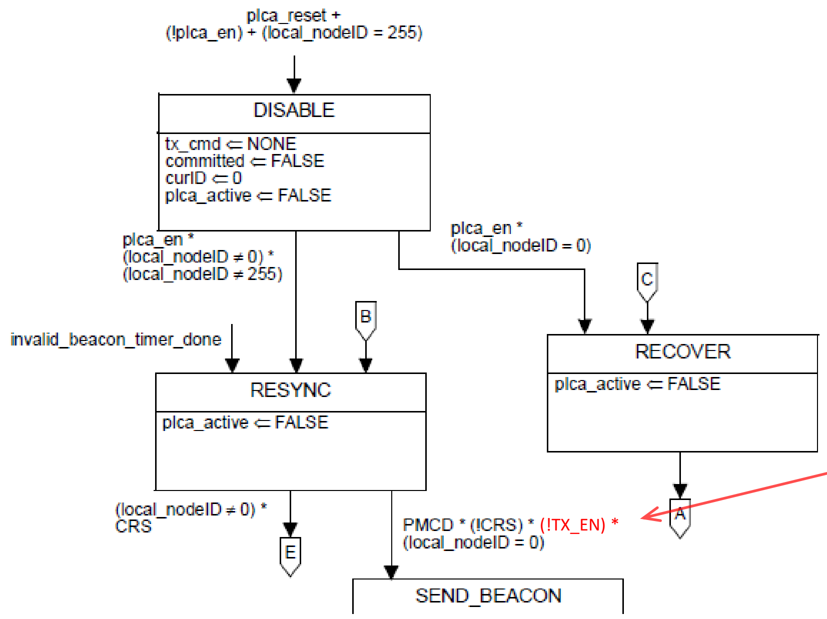


NORMAL
packetPending ⇐ FALSE
IF CRS THEN
CARRIER_STATUS ⇐ CARRIER_ON
ELSE
CARRIER_STATUS ⇐ CARRIER_OFF
END
TXD ⇐ plca_txd
TX_EN ⇐ plca_txen
TX_ER ⇐ plca_txer
IF COL THEN
SIGNAL_STATUS ⇐ SIGNAL_ERROR
ELSE
SIGNAL_STATUS ⇐ NO_SIGNAL_ERROR
END
packetPending
CARRIER_STA
SIGNAL_STAT
TXD ⇐ ENCOI
TX_EN ⇐ FAL
TX_ER ⇐ ENC
a ⇐ 0
b ⇐ 0
MCD * (!CRS)
ELSE
!receiving) * (!plca_txen)
plca_en * (!plca_reset) * (plca_status = OK)
IDLE
packetPending ⇐ FALSE
CARRIER_STATUS ⇐ CARRIER_OFF
SIGNAL_STATUS ⇐ NO_SIGNAL_ERROR
TXD ⇐ ENCODE_TXD(tx_cmd_sync)
TX_EN ⇐ FALSE
TX_ER ⇐ ENCODE_TXER(tx_cmd_sync)

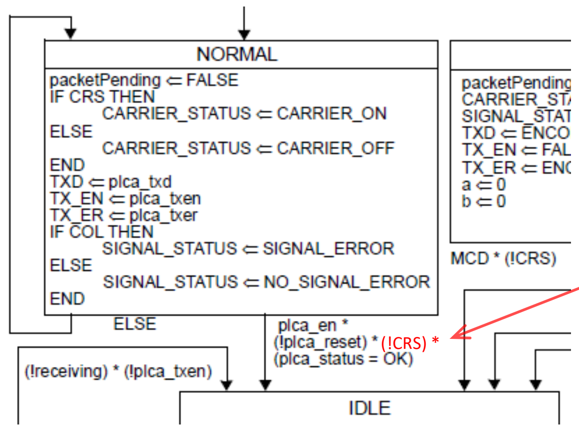
PLCA DATA State Diagram

- The problem shows up if the aforementioned frame is sent a few bit times before the PLCA Control state diagram enters the RESYNC state
- In such a case, the CRS would not yet be asserted due to the MII and PHY latencies, therefore letting the PLCA Control state diagram go to SEND_BEACON state
  – This makes the PLCA Data state diagram move to IDLE state (because plca_active = TRUE, plca_status = OK)
- In IDLE state, the frame data coming from the MAC is overwritten by tx_cmd being set to BEACON
  – The frame gets corrupted at the MII (!)

# #1: fix



**PLCA Control State Diagram, part a**



**PLCA DATA State Diagram, part a**

- A simple solution to prevent this race condition is to inhibit the PLCA Control State Diagram from entering the SEND_BEACON state while the DATA State Diagram is in the process of sending a frame.
- Additionally, the transition from NORMAL to IDLE in the PLCA DATA State Diagram should be conditioned by CRS being low

## #2: Data corruption at the end of an aborted PLCA burst

## #2: background

- When PLCA burst mode is activated (max_bc > 0), the PLCA RS waits the MAC to send a new frame for the duration of "burst_timer" before moving to the next transmit opportunity.
  - During this time, a COMMIT is conveyed to the PHY to "keep" the current TO by extending the carrier (CRS = TRUE).
- If the MAC conveys a new frame while the burst_timer is active, the burst succeeds and the PLCA RS forwards the frame to the PHY via MII (appended to the COMMIT)
- else (i.e. the MAC has no further frames to send), the PLCA RS aborts the burst and moves to the next TO
  - If the MAC initiates a frame transmission (plca_txen = TRUE) concurrently with the burst_timer expiration, the PLCA RS **is supposed** to just act as normal, delaying the frame until the next TO…
    - BUT there's a race condition in the state diagrams!

# #2: problem

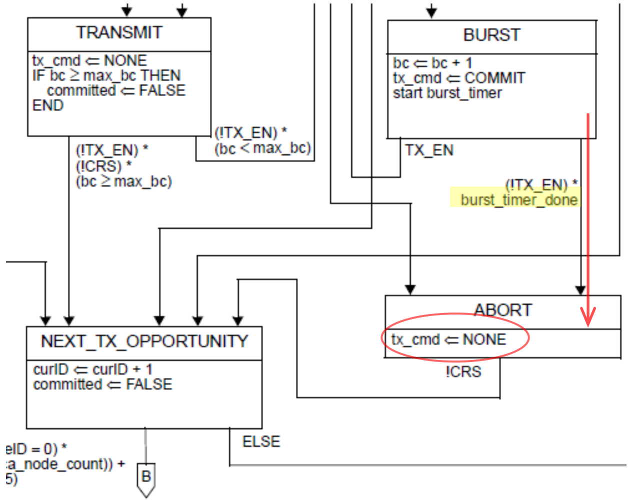
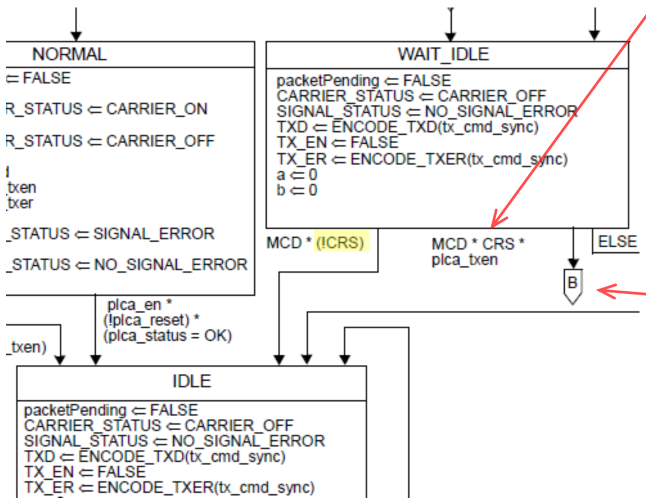


Figure 148–4—PLCA Control state diagram, part b



NORMAL
WAIT_IDLE
packetPending ⇐ FALSE
CARRIER_STATUS ⇐ CARRIER_OFF
SIGNAL_STATUS ⇐ NO_SIGNAL_ERROR
TXD ⇐ ENCODE_TXD(tx_cmd_sync)
TX_EN ⇐ FALSE
TX_ER ⇐ ENCODE_TXER(tx_cmd_sync)
a ⇐ 0
b ⇐ 0
MCD * (!CRS)
MCD * CRS * plca_txen
ELSE
B
plca_en * (!plca_reset) * (plca_status = OK)
IDLE
packetPending ⇐ FALSE
CARRIER_STATUS ⇐ CARRIER_OFF
SIGNAL_STATUS ⇐ NO_SIGNAL_ERROR
TXD ⇐ ENCODE_TXD(tx_cmd_sync)
TX_EN ⇐ FALSE
TX_ER ⇐ ENCODE_TXER(tx_cmd_sync)

PLCA DATA State Diagram, part a

- When the burst_timer expires, the PLCA Control State Diagram goes from BURST to ABORT state.
- Meanwhile, the PLCA Data state diagram is waiting in WAIT_IDLE state for the MAC to send a new frame (plca_txen = TRUE) or the burst to be aborted (CRS going low due to tx_cmd being set to NONE)
- However, if the MAC starts conveying a frame a few bit times after the burst was aborted (before MCD), the PLCA Data state diagram would take the path of the successful burst "B" while the PLCA Control state diagram already aborted the burst (!)
  – This may lead to the frame being corrupted and to unwanted collision on the line

# #2: fix

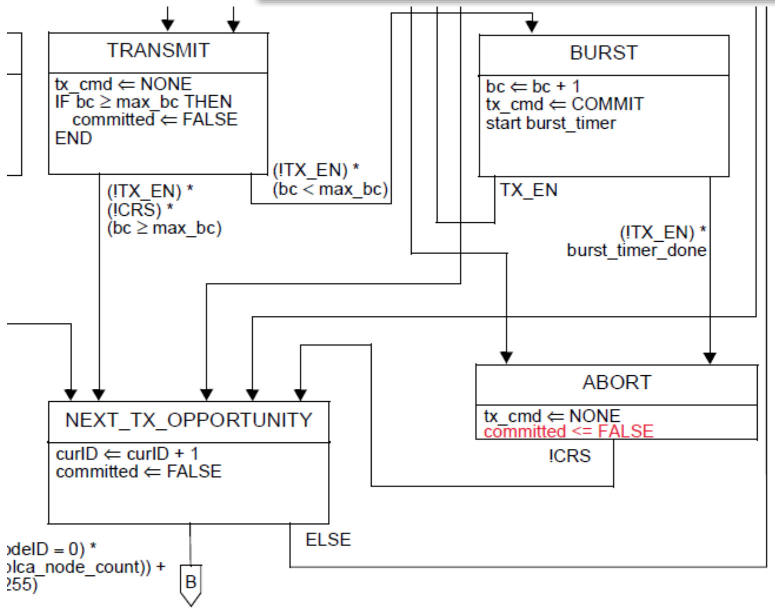
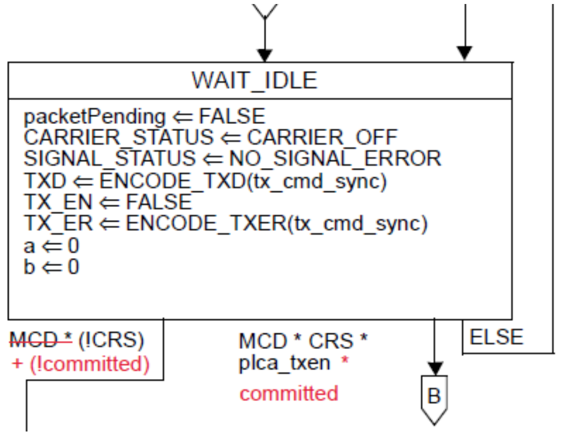


**Figure 148–4—PLCA Control state diagram, part b**



WAIT_IDLE
packetPending ⇐ FALSE
CARRIER_STATUS ⇐ CARRIER_OFF
SIGNAL_STATUS ⇐ NO_SIGNAL_ERROR
TXD ⇐ ENCODE_TXD(tx_cmd_sync)
TX_EN ⇐ FALSE
TX_ER ⇐ ENCODE_TXER(tx_cmd_sync)
a ⇐ 0
b ⇐ 0
~~MCD *~~ (!CRS)
+ (!committed)
MCD * CRS *
plca_txen *
committed
ELSE
B

**PLCA DATA State Diagram, part a**

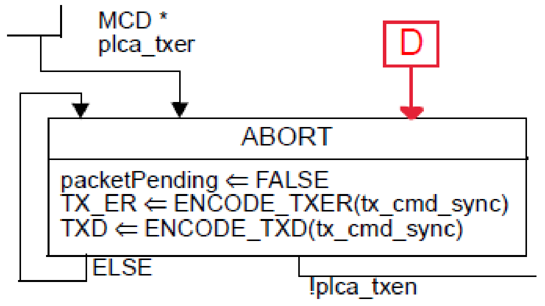
- The fix is to avoid the race condition by removing the synchronization on the MII clock (MCD) which is not required in that state anyway.
- Then, we can synchronize on the “committed” variable, instead.

## #3: frame aborted by the upper layers (plca_txer = 1)

## #3: background

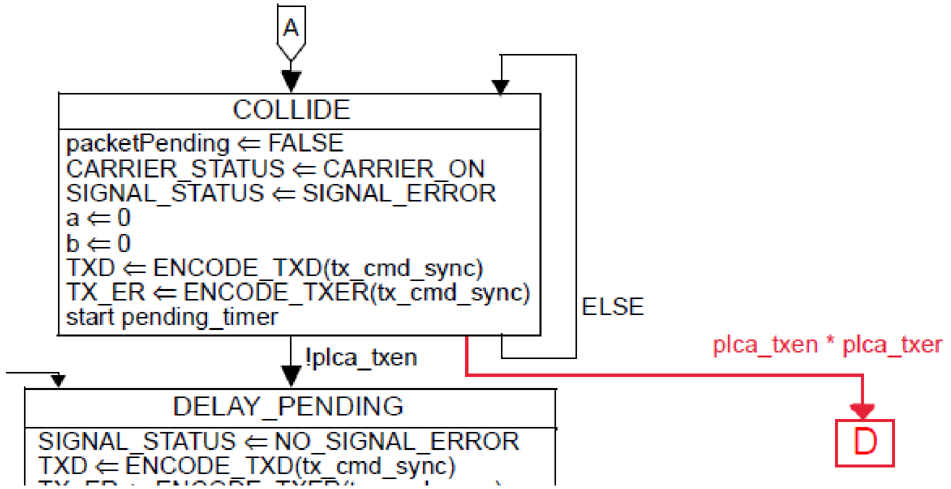
- The PLCA RS detects collisions by both monitoring the COL signal from the PHY and checking transmissions vs transmit opportunities
- In the latter case, the PLCA RS relies on the MAC to backoff and make a new attempt at sending the frame after the carrier indication goes off
- If the upper layers aborted the frame concurrently with the PLCA RS detecting a collision (very rare race), the next transmit opportunity may be skipped, leading to wasted bandwidth and/or multiple collisions

## #3: solution



MCD * plca_txer
D
ABORT
packetPending ⇐ FALSE
TX_ER ⇐ ENCODE_TXER(tx_cmd_sync)
TXD ⇐ ENCODE_TXD(tx_cmd_sync)
ELSE
!plca_txen

**PLCA DATA State Diagram, part a**



A
COLLIDE
packetPending ⇐ FALSE
CARRIER_STATUS ⇐ CARRIER_ON
SIGNAL_STATUS ⇐ SIGNAL_ERROR
a ⇐ 0
b ⇐ 0
TXD ⇐ ENCODE_TXD(tx_cmd_sync)
TX_ER ⇐ ENCODE_TXER(tx_cmd_sync)
start pending_timer
ELSE
!plca_txen
plca_txen * plca_txer
D
DELAY_PENDING
SIGNAL_STATUS ⇐ NO_SIGNAL_ERROR
TXD ⇐ ENCODE_TXD(tx_cmd_sync)

**PLCA DATA State Diagram, part b**

- A simple fix is to check the already existing plca_txer variable while the PLCA Data State Diagram is in the COLLIDE state.
- If plca_txer is set, just go into ABORT state, letting the PLCA RS serve the next TO without setting packetPending to TRUE

## CONCLUSIONS

## Conclusions

- The current definition of the PLCA state diagrams hides unwanted (rare) race conditions leading to corrupted frames and/or performance loss
- This presentation shows simple fixes that don't break interoperability with systems implementing what's currently specified in Clause 148.
- I suggest 802.3da to adopt these fixes as enhancements over 802.3cg

# THANK YOU