Pseudocode update in 802.1CB-2017 format
Fejes Ferenc, Balázs Varga – Ericsson
2026-10-30

CBec amendment refines the sequence generation/recovery functions. This contribution contains an initial pseudocode update in 802.1CB-2017 format. Diffs from current P802.1CB-Rev/D1.2 are highlighted with color. Green means new added code. ~~Red~~ means deleted code and Blue highlights new management variables.

Note: this pseudocode does not contain all the discussed functional additions. For example it misses the PostResetTimeGuard related code, definition of new variables, usage of the reset flag, etc. The structure of the proposed pseudocode is up to discussion, there might be room for improvement or simplification.

BACKWARD COMPATIBILITY

The initialization sequence space related operations proposed by this document can be disabled with the UseInitSeqNum management variable. By setting this variable to **false** the behavior of the sequence generator or the recovery behaves as defined in P802.1CB-Rev/D1.2. The initialization related operations can be enabled with setting the variable to **true** value.

SEQUENCE GENERATION

```
-------------------------------------------------------------------------------
7.4.1.3 SequenceGenerationReset
-------------------------------------------------------------------------------
void SequenceGenerationReset () {
    GenSeqNum = 0;
    InitGenSeqNum = InitGenSeqStart;
    SendResetFlagNum = 3;  // it may be frerSeqRcvyHistoryLength ??
    frerCpsSeqGenResets = frerCpsSeqGenResets + 1;
}

-------------------------------------------------------------------------------
7.4.1.4 SequenceGenerationAlgorithm
-------------------------------------------------------------------------------

void SequenceGenerationAlgorithm () {
    unsigned int sequence_number = GenSeqNum;
    unsigned int init_flag = 0;
    unsigned int reset_flag = 0;
    ~~if (GenSeqNum >= (GenSeqSpace - 1))~~
    // Using InitSeqNumSpace
    if (UseInitSeqNum && InitGenSeqNum <= (GenSeqSpace - 1)) {
        sequence_number = InitGenSeqNum;
        init_flag = 1;
        InitGenSeqNum = InitGenSeqNum + 1;
        if (SendResetFlagNum > 0) {
            reset_flag = 1;
            SendResetFlagNum = SendResetFlagNum - 1;
        }
    }
    // Using normal SeqNumSpace
    else if (GenSeqNum >= (GenSeqSpace - 1))
        GenSeqNum = 0;
    else
        GenSeqNum = GenSeqNum + 1;
    ~~SEND_DATA;  // Pass packet & sequence_number subparameter down stack~~
    SEND_DATA;  // Pass packet & sequence_number, init_flag, reset_flag down
}
```

SEQUENCE RECOVERY

7.4.3.1 Events for sequence recovery

---

c) RECOVERY_TIMEOUT: The event that occurs when the RemainingTicks (7.4.3.2.4) variable reaches 0. It triggers the execution of the ~~SequenceRecoveryReset~~ SetVariablesDueTimeout routine (~~7.4.3.3~~ X.X.X.X). The same event occurs when InitRemainingTicks reaches 0, and triggers the execution of the InitSetVariablesDueTimeout routine (X.X.X.X).

7.4.3.3 SequenceRecoveryReset

```
void SequenceRecoveryReset () {
    if (frerSeqRcvyAlgorithm == Vector_Alg) {
        int i;
        RecovSeqNum = RecovSeqSpace - 1;
        for (i = 0; i < frerSeqRcvyHistoryLength; i = i + 1)
            SequenceHistory[i] = 0;  // Set all bits 0 (packet not seen)
        SuppressLostUntilPacket = true;
        InitSequenceRecoveryReset();
    }
    frerCpsSeqRcvyResets = frerCpsSeqRcvyResets + 1;
    TakeAny = true;
}
```

7.4.3.6 ShiftSequenceHistory

```
void ShiftSequenceHistory (int newZeroValue) {
    int i;
    if (0 == SequenceHistory[frerSeqRcvyHistoryLength - 1])
    if (SuppressLostUntilPacket) {
        if (1 == SequenceHistory[frerSeqRcvyHistoryLength - 1])
            SuppressLostUntilPacket = false;  // a 1 has crossed the window
    }
    else if (0 == SequenceHistory[frerSeqRcvyHistoryLength - 1])
        frerCpsSeqRcvyLostPackets = frerCpsSeqRcvyLostPackets + 1;
    for (i = frerSeqRcvyHistoryLength - 1; i != 0; i = i - 1)
        SequenceHistory[i] = SequenceHistory[i - 1];
    SequenceHistory[0] = newZeroValue;
}
```

X.X.X.X SelectVectorRecovery

```
void SelectVectorRecovery () {
    unsigned int init_flag;
    if (UseInitSeqNum && init_flag)
        InitVectorRecoveryAlgorithm();
    else
        VectorRecoveryAlgorithm();
}
```

```
-------------------------------------------------------------------------------
X.X.X.X InitSequenceRecoveryReset
-------------------------------------------------------------------------------

void InitSequenceRecoveryReset () {
    int i;
    InitRecovSeqNum = InitRecovSeqStart - 1;
    for (i = 0; i < frerSeqRcvyHistoryLength; i = i + 1)
        InitSequenceHistory[i] = 0;  // Set all bits 0 (packet not seen)
    InitTakeAny = true;
    InitSuppressLostUntilPacket = true;
}


-------------------------------------------------------------------------------
X.X.X.X SetVariablesDueTimeout
-------------------------------------------------------------------------------

void SetVariablesDueTimeout () {
    if (frerSeqRcvyAlgorithm == Vector_Alg) {
        int i;
        RecovSeqNum = RecovSeqSpace - 1;
        for (i = 0; i < frerSeqRcvyHistoryLength; i = i + 1) {
            if (false == SuppressLostUntilPacket && 0 == SequenceHistory[i])
                frerCpsSeqRcvyLostPackets = frerCpsSeqRcvyLostPackets + 1;
            SequenceHistory[i] = 0;  // Set all bits 0 (packet not seen)
        }
        SuppressLostUntilPacket = true;
    }
    TakeAny = true;
    frerCpsSeqRcvyTimeouts = frerCpsSeqRcvyTimeouts + 1;
}


-------------------------------------------------------------------------------
X.X.X.X InitSetVariablesDueTimeout
-------------------------------------------------------------------------------

void InitSetVariablesDueTimeout () {
    int i, pending;
    pending = 0;  // zeros seen since the last 1
    for (i = 0; i < frerSeqRcvyHistoryLength; i = i + 1) {
        if (0 == InitSequenceHistory[i])
            pending = pending + 1;
        else {  // a 1: the zeros before it are holes
            frerCpsSeqRcvyLostPackets = frerCpsSeqRcvyLostPackets + pending;
            pending = 0;
        }
    }
    if (false == InitSuppressLostUntilPacket)
        frerCpsSeqRcvyLostPackets = frerCpsSeqRcvyLostPackets + pending;
    InitSequenceRecoveryReset();
    frerCpsSeqRcvyInitTimeouts = frerCpsSeqRcvyInitTimeouts + 1;
}
```

```
-------------------------------------------------------------------------------
X.X.X.X InitShiftSequenceHistory
-------------------------------------------------------------------------------

void InitShiftSequenceHistory (int newZeroValue) {
    int i;
    if (InitSuppressLostUntilPacket) {
        if (1 == InitSequenceHistory[frerSeqRcvyHistoryLength - 1])
            InitSuppressLostUntilPacket = false;  // a 1 has crossed the window
    }
    else if (0 == InitSequenceHistory[frerSeqRcvyHistoryLength - 1])
        frerCpsSeqRcvyLostPackets = frerCpsSeqRcvyLostPackets + 1;
    for (i = frerSeqRcvyHistoryLength - 1; i != 0; i = i - 1)
        InitSequenceHistory[i] = InitSequenceHistory[i - 1];
    InitSequenceHistory[0] = newZeroValue;
}


-------------------------------------------------------------------------------
X.X.X.X InitVectorRecoveryAlgorithm
-------------------------------------------------------------------------------

void InitVectorRecoveryAlgorithm () {
    // Check that sequence number is present in the packet
    unsigned int sequence_number;
    if (sequence_number == frerSeqRcvyInvalidSequenceValue) {
        frerCpsSeqRcvyTaglessPackets = frerCpsSeqRcvyTaglessPackets + 1;
        if (frerSeqRcvyTakeNoSequence) {
            frerCpsSeqRcvyPassedPackets = frerCpsSeqRcvyPassedPackets + 1;
            frerCpSeqRcvyPassedPackets = frerCpSeqRcvyPassedPackets + 1;
            InitRemainingTicks =
                    ((frerSeqRcvyResetMSec*TicksPerSecond)+999)/1000;
            PRESENT_DATA;
        } else {
            frerCpsSeqRcvyDiscardedPackets =
                    frerCpsSeqRcvyDiscardedPackets + 1;
            frerCpSeqRcvyDiscardPackets = frerCpSeqRcvyDiscardPackets + 1;
        }
        return;
    }
    // Compute signed difference in the linear init space (no wrap).
    int delta = (int)sequence_number - (int)InitRecovSeqNum;
    // Not an init number: rogue, also as the first packet of a phase.
    if (sequence_number < InitRecovSeqStart ||
        sequence_number >= RecovSeqSpace  )
    {
        // Packet is out-of-range. Count and discard it.
        frerCpsSeqRcvyRoguePackets = frerCpsSeqRcvyRoguePackets + 1;
        frerCpSeqRcvyDiscardPackets = frerCpSeqRcvyDiscardPackets + 1;
        // Reset timer if working on an individual Stream
        if (frerSeqRcvyIndividualRecovery)
            InitRemainingTicks =
                    ((frerSeqRcvyResetMSec * TicksPerSecond) + 999) / 1000;
    }
    // After reset, accept any packet
    else if (InitTakeAny) {
        InitTakeAny = false;
        InitSequenceHistory[0] = 1;  // Shift, adding a "seen" bit
```

```
        InitRecovSeqNum = sequence_number;
        frerCpsSeqRcvyPassedPackets = frerCpsSeqRcvyPassedPackets + 1;
        frerCpSeqRcvyPassedPackets = frerCpSeqRcvyPassedPackets + 1;
        InitRemainingTicks =
                ((frerSeqRcvyResetMSec * TicksPerSecond) + 999) / 1000;
        PRESENT_DATA;
        return;
    }
    else if (delta > frerSeqRcvyHistoryLength ||
             delta <= -frerSeqRcvyHistoryLength  )
    {
        // Packet is out-of-range. Count and discard it.
        frerCpsSeqRcvyRoguePackets = frerCpsSeqRcvyRoguePackets + 1;
        frerCpSeqRcvyDiscardPackets = frerCpSeqRcvyDiscardPackets + 1;
        // Reset timer if working on an individual Stream
        if (frerSeqRcvyIndividualRecovery)
            InitRemainingTicks =
                    ((frerSeqRcvyResetMSec * TicksPerSecond) + 999) / 1000;
    }
    else if (delta <= 0) {
        // Packet is old and in InitSequenceHistory; have we seen it before?
        if (0 == InitSequenceHistory[-delta]) {
            // Packet has not been seen. Take it.
            InitSequenceHistory[-delta] = 1;
            frerCpsSeqRcvyOutOfOrderPackets =
                    frerCpsSeqRcvyOutOfOrderPackets + 1;
            frerCpsSeqRcvyPassedPackets = frerCpsSeqRcvyPassedPackets + 1;
            frerCpSeqRcvyPassedPackets = frerCpSeqRcvyPassedPackets + 1;
            InitRemainingTicks =
                    ((frerSeqRcvyResetMSec * TicksPerSecond) + 999) / 1000;
            PRESENT_DATA;
        } else {
            // Packet has been seen. Do not forward. Count the discard.
            frerCpsSeqRcvyDiscardedPackets =
                    frerCpsSeqRcvyDiscardedPackets + 1;
            frerCpSeqRcvyDiscardPackets = frerCpSeqRcvyDiscardPackets + 1;
            // Reset timer if working on an individual Stream
            if (frerSeqRcvyIndividualRecovery)
                InitRemainingTicks =
                        ((frerSeqRcvyResetMSec * TicksPerSecond) + 999) / 1000;
        }
    }
    else {
        // Packet is not too far ahead of the one we want.
        // Packet is out-of-order unless it directly follows InitRecovSeqNum
        if (delta != 1)
            frerCpsSeqRcvyOutOfOrderPackets =
                    frerCpsSeqRcvyOutOfOrderPackets + 1;
        // Shift the history until bit 0 refers to sequence_number.
        while (0 != (delta = delta - 1))
            InitShiftSequenceHistory(0);  // Shift, adding a "not seen" bit
        InitShiftSequenceHistory(1);  // Shift, adding a "seen" bit
        InitRecovSeqNum = sequence_number;
        frerCpsSeqRcvyPassedPackets = frerCpsSeqRcvyPassedPackets + 1;
        frerCpSeqRcvyPassedPackets = frerCpSeqRcvyPassedPackets + 1;
        InitRemainingTicks =
                ((frerSeqRcvyResetMSec * TicksPerSecond) + 999) / 1000;
        PRESENT_DATA;
    }
}
```

P802.1CBec SCENARIO SIMULATOR

To help the brainstorming and the discussions, we created a simulator to try out different P802.1CBec scenarios. This is done by a discrete event simulation of the vector recovery and the proposed improved version with the init sequence number space. With this tool one can compare how would CB behave (counters, state) for the same input sequences, which does not care about the init flag.

The simulator is a webapp; it is available here:

https://www.ieee802.org/1/files/public/docs2026/ec-fejes-varga-FRER-simulator-1026-v01.html

Advantages of our proposed solution

1. All counters defined in the standard remain consistent both during the initialization phase and during the transition to the normal phase.

2. The proposal is backward compatible; the management variable setting UseInitSeqNum=false disables the init operating mode. In an implementation, this consumes no additional computational resources compared to normal operation.
2.1. On the generator side as well, the management variable UseInitSeqNum=false disables the init sequence number space.

3. The VectorRecoveryAlgorithm function is not modified. Existing verifications and conformance tests remain valid.

4. Our proposal can also be implemented such that the existing, unmodified VectorRecoveryAlgorithm function is executed on the state space belonging to the init sequence number space (InitTakeAny, InitSequenceHistory, InitRemainingTicks).