Simulating CCSDS L7 Packets and L2 Framing: Plan & Pseudocode

1 Key Correction (Before You Simulate)

- If a Space Packet merely **spans multiple TM/AOS frames at L2**, keep the Space-Packet **Sequence Flags = 11 (standalone)**. L2 handles spanning via the **First Header Pointer (FHP)** and frame packing.
- Use Space-Packet segmentation flags (`01` first, `00` continuation, `10` last) **only** when you intentionally split one L7 message into *multiple Space Packets at L7*. In that case, the **Sequence Count increments for every segment** (as for any packet).

2 What to Simulate (and Why)

1. **L7 Packet Generator (per APID/app)**: Build complete CCSDS Space Packets with primary header (Version, Type, SecHdrFlag, APID, SeqFlags, SeqCount, Length), optional secondary header (e.g., time tag), user data, and optional L7 MIC (CRC-32C) at the end of the user data. *Default: SeqFlags = 11; SeqCount increments per packet.*
2. **L2 Frame Packer** (e.g., 1115-byte TM data field): Pack the byte stream of Space Packets into TM/AOS frames, computing and writing **FHP**; carry bytes across frames as needed. This validates spanning without touching L7 flags.
3. **CADU Maker (optional in Python)**: Prepend **ASM** (e.g., `1A CF FC 1D`), apply **randomizer** (exclude ASM) and **FEC** (RS+Conv or LDPC/Turbo) if you want a full offline pipeline. In most workflows, you let **GRC** perform these.
4. **L2 Reassembler (RX-side)**: Use FHP to find packet starts and reassemble complete Space Packets across frames; emit whole packets to L7.
5. **L7 Verifier**: On each complete Space Packet, verify APID, sequence continuity, and optional L7 MIC (CRC-32C over user data).

3 Pseudocode Sketches (Language-Agnostic)

L7 Packet Generator

```
for each app in test_apps:
    apid = app.apid
    for n in range(num_packets):
        sec_hdr  = build_secondary_header(time_mode)       # fixed or now()
        user     = build_user_data(app, n)                 # deterministic bytes
        if use_mic:
            mic  = crc32c(user)                            # over user data only
            user = user || mic_be                          # big-endian 4B
        # Primary header fields
        flags     = 0b11                                   # standalone (no L7 segmentation)
        seq_count = next_seq_count(apid)                   # increments per packet
        length    = len(sec_hdr || user) - 1               # bytes after 6B header minus 1
        hdr       = pack_primary_header(ver=0, type=app.type,
                                        shf=(sec_hdr!=None), apid=apid,
                                        flags=flags, seq=seq_count, length=length)
        spp = hdr || sec_hdr || user
        emit(spp)
```

L2 Frame Packer (e.g., 1115-byte data field)

```
FRAME_DATA_BYTES = 1115
init new_frame
fhp = compute_next_header_offset_or_no_header()
while spp_bytes_remaining:
    copy as many SPP bytes as fit in remaining_frame_space
    if SPP completes and another SPP header will start in this frame:
        update fhp to that offset
    if frame full or no more input this instant:
        frame = make_tm_frame(fhp, frame_data_bytes)
        emit(frame)
        init new_frame; fhp = compute_next_header_offset_or_no_header()
```

CADU Maker (optional; usually GRC)

```
# If done offline:
coded   = fec_encode( randomize(frame_bytes, exclude_asm=True) )
cadu    = ASM || coded
emit(cadu)
```

L2 RX Reassembler (frame -¿ packets)

```
for each frame:
    k = fhp(frame)                  # header offset or "no header here"
    if k indicates header start:
        start new SPP assembly at k
    append frame payload into current SPP buffer(s)
    if collected_bytes == (6 + length+1) for a given SPP:
        emit complete_spp
```

L7 Verifier

```
on complete_spp:
```

```
(apid, flags, seq, length) = parse_primary_header(spp[0..5])
assert flags == 0b11 unless intentionally L7-segmented
user_data, mic_opt = split_user_and_mic(spp)
if use_mic:
    assert crc32c(user_data) == mic_opt
assert seq continuity per apid
log csv row (apid, seq, mic_ok, continuity_ok, ...)
```

4 What You Will See in the First 64 Bytes

- **L7 (Space Packet):** First 64 bytes = `PrimaryHeader (6B)` + `SecondaryHeader (S)` + start of **user data**. This is true even if the total packet is larger than the frame data field (e.g., 1115 B). No scrambling/coding; human-readable patterns persist.

- **L2/CADU:** First 64 bytes = **ASM** (e.g., `1A CF FC 1D`) + randomized/FEC-encoded frame bytes. These look noise-like and will not resemble the L7 header/payload. Middle frames (when spanning) may have FHP = "no header".

5 When to Use Space-Packet Segmentation Flags

Use `01/00/10` (*first/continuation/last*) only when you choose to **split one application message into multiple Space Packets at L7**. In that case:

- Generate multiple Space Packets (each with its own primary header).

- **Sequence Count increments** for every segment (as for any new packet).

- L2 may still span any of those packets across multiple frames; FHP handles that independently.

If your packet is simply larger than the frame payload and you rely on L2 spanning, **do not change** the Space-Packet flags; keep `11` (standalone) and rely on FHP.

6 Bottom Line (Execution Plan)

1. Implement the L7 generator and verifier with parameters for APID, time mode (fixed vs. now), MIC on/off, and payload size/pattern.

2. Either: (a) feed Space Packets directly into GRC to perform framing/randomization/FEC and capture CADUs; or (b) implement a light frame packer offline to validate FHP logic, then still use GRC for coding.

3. Record artifacts: L7 golden packet (binary/hex), L7 MIC (if used), CADU first-64 bytes (from post-ASM), per-packet CSV logs, and a short summary (packets, MIC fails, seq gaps, pps).