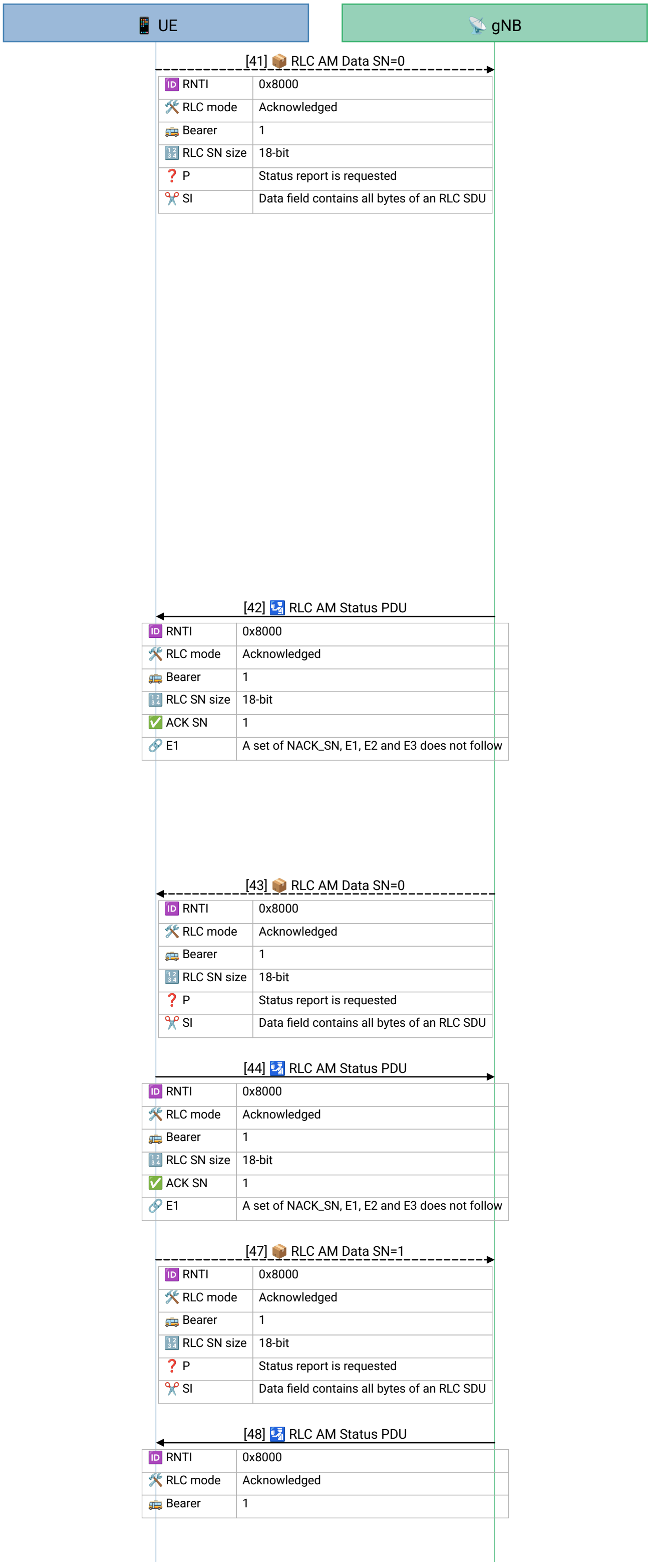


[Frame 41] Where this bearer came from. This diagram opens on a Radio Link Control (RLC) entity that did not exist a fraction of a second earlier. Frame 38 of the capture — on the signaling bearer, so it is not on this diagram — carried an RRC Reconfiguration that added **Data Radio Bearer (DRB) 1** on logical-channel ID 4 and configured its RLC in Acknowledged Mode with:

- `sn-FieldLength` — size18, in both directions
- `t-PollRetransmit` — ms80 (uplink)
- `pollPDU` / `pollByte` — p32768 / kB750 (uplink)
- `t-Reassembly` — ms80 (downlink)
- `t-StatusProhibit` — ms30 (downlink)

Everything you are about to read follows from those five lines, starting with this frame.

The first PDU. This is the first protocol data unit (PDU) the new entity puts on the air. `SN=0`, and the sequence-number field is **18 bits wide** — the packet trace confirms bearer type DRB, bearer id 1, sequence-number length 18. RLC numbers *SDUs* — service data units, the PDCP packets handed down to it — not bytes and not transmissions, and a new entity starts that count at zero with nothing in its retransmission buffer. `SI` = "all bytes of an RLC SDU" (the 2-bit Segmentation Info field, value 0) says this PDCP packet fitted in one PDU, and P — the poll bit — obliges the gNB to send a status report straight back.

The wider counter is not decoration. Eighteen bits gives a 262144-value sequence space against 4096 for the 12-bit form used on signaling bearers, which is what a high-throughput bearer needs so its transmit window never wraps around data that is still unacknowledged.

[Frame 42] The first acknowledgment, 0.8 ms later. A STATUS PDU is RLC's only control message and it is 3 bytes long. `ACK_SN=1` does not mean "SDU 1 arrived" — it means *"the next SDU I am still waiting for is number 1"*, so everything below 1, namely SN 0, is safely in. `E1=0` says no NACK block follows — a NACK, or negative acknowledgment, is how a receiver names a specific sequence number it never got — so nothing is missing.

Note the `ACK_SN` field is itself 18 bits wide here: a STATUS PDU's format follows the entity's configured sequence-number length, so the acknowledgment is shaped by the same `sn-FieldLength` line that shaped frame 41.

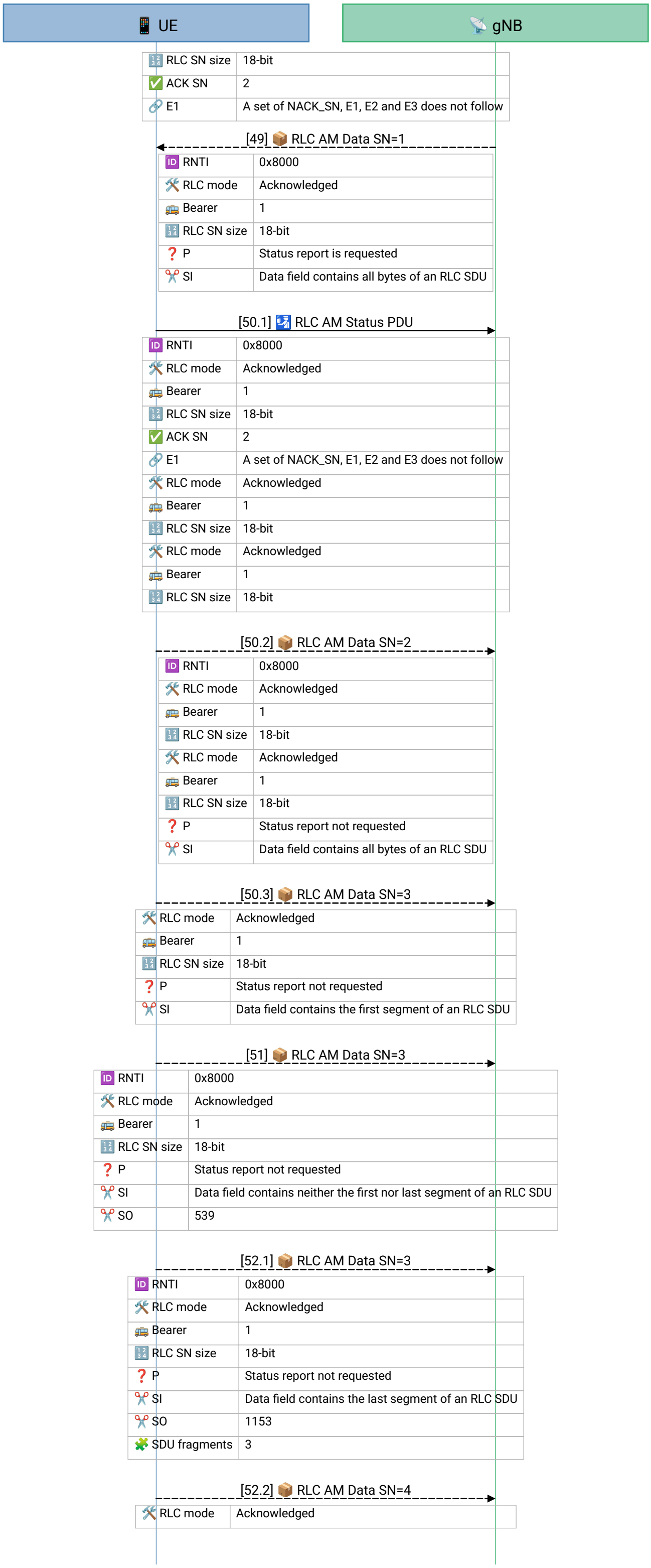
Fix this in your head now: **there is not one NACK in this entire capture.** Every status on this diagram is a pure cumulative acknowledgment.

[Frame 43] The downlink has its own counter. The gNB sends *its* `SN=0`. That is not a repeat of frame 41 — an AM bearer is two independent transmitting entities, one per direction, each with its own sequence numbers, poll timer and retransmission buffer. Uplink SN 0 and downlink SN 0 are unrelated packets.

[Frame 44] Same acknowledgment, eighteen times slower. `ACK_SN=1` confirms the downlink SN 0 of frame 43, but it takes 14.2 ms where the gNB answered in 0.8 ms. RLC is not the bottleneck: the gNB schedules its own downlink immediately, while the UE cannot transmit even 3 bytes until it is granted uplink airtime. Every uplink status on this diagram costs 11 to 20 ms for that reason.

[Frame 47] Uplink `SN=1`, `SI` = all bytes, poll set. Traffic is still light enough that one SDU makes one PDU.

[Frame 48] `ACK_SN=2` — uplink SN 0 and 1 delivered, so the transmitter can drop both from its retransmission buffer.



[Frame 49] Downlink SN=1 , poll set.

[Frame 50.1] Three RLC PDUs in one radio transmission. The uplink buffer has filled, and MAC packs everything that fits into the single grant it was given. The STATUS PDU comes out first: ACK_SN=2 clears the downlink SN 0 and 1 and answers frame 49's poll.

Piggybacking control onto a data grant is the normal case, not an optimisation — RLC hands MAC whatever it has and MAC fills the transport block. The diagram splits the three PDUs into separate arrows so each can be read on its own.

[Frame 50.2] Uplink SN=2 , a complete 64-byte SDU — and P is **not** set, the first data PDU on this bearer without a poll. More data is queued behind it, so RLC withholds the poll. This is the rule that governs the rest of the diagram, and it comes straight from the configuration in frame 41: with `pollPDU` at p32768 and `pollByte` at kB750, the counter-based poll triggers will effectively never fire, so a poll happens only when the transmit buffer is about to run dry (TS 38.322 §5.3.3.2).

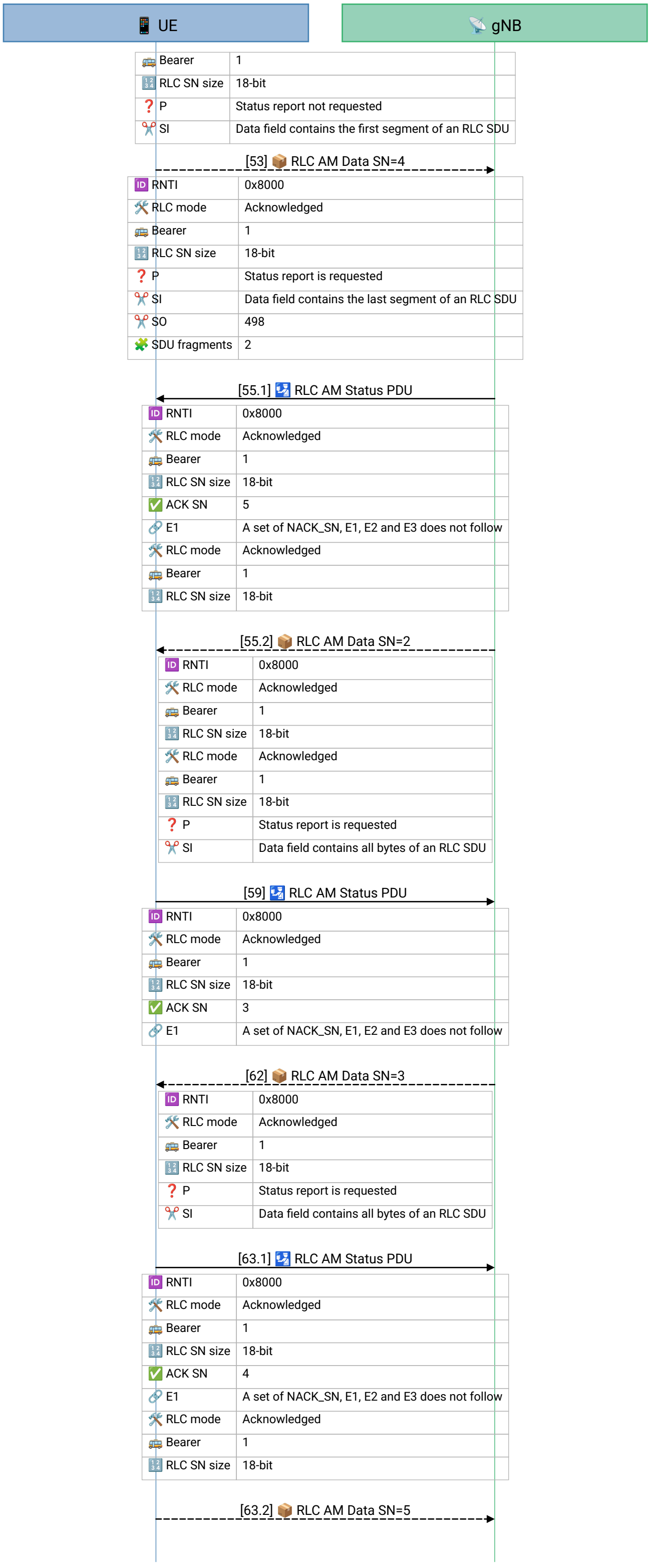
[Frame 50.3] Uplink SN=3 , SI = "first segment of an RLC SDU" (value 1), still no poll. A 1264-byte SDU starts here, and the grant has room for only 539 of its bytes after the 3-byte header. Segmentation is never a property of the data — it is a property of the grant that happened to be available.

[Frame 51] SN=3 continues: SI = "neither the first nor last segment" (value 3), SO=539 . The Segment Offset is a byte count from the start of the original SDU, and 539 is exactly where frame 50 stopped. The header is 5 bytes here — 3 for the 18-bit sequence-number fields plus 2 for SO , which a non-first segment must carry — so 619 bytes on the wire mean 614 bytes of payload, ending at offset 1153.

[Frame 52.1] SN=3 completes: SI = "last segment" (value 2), SO=1153 . Wireshark reassembles 3 fragments totalling 1264 bytes, contributed as 539 + 614 + 111 — the segments tile the SDU with no gap and no overlap.

All three segments carry the same sequence number. The number belongs to the SDU; segmentation is described entirely by SI and SO . Still no poll, because as the next arrow shows there is more in the buffer.

[Frame 52.2] Uplink SN=4 starts in the same transport block: SI = first segment, 498 bytes of a 514-byte SDU. Two different SDUs are mid-flight in the same frame, told apart only by their sequence numbers.



[Frame 53] SN=4 completes with SO=498 and just 16 bytes of payload — a 2-fragment, 514-byte reassembly. **Now** the poll goes up: the transmit buffer is finally empty, and the UE wants the whole burst confirmed at once.

[Frame 55.1] One status for five SDUs. ACK_SN=5 acknowledges uplink SN 0 through 4 in a single 3-byte control PDU — including SN 3 and SN 4, which between them cost five separate transmissions across frames 50 to 53, answered 2.8 ms after the poll. This is what the poll discipline buys: hold the poll until the buffer drains, and one status closes the entire burst. ARQ (automatic repeat request) state is per SDU, not per transmission.

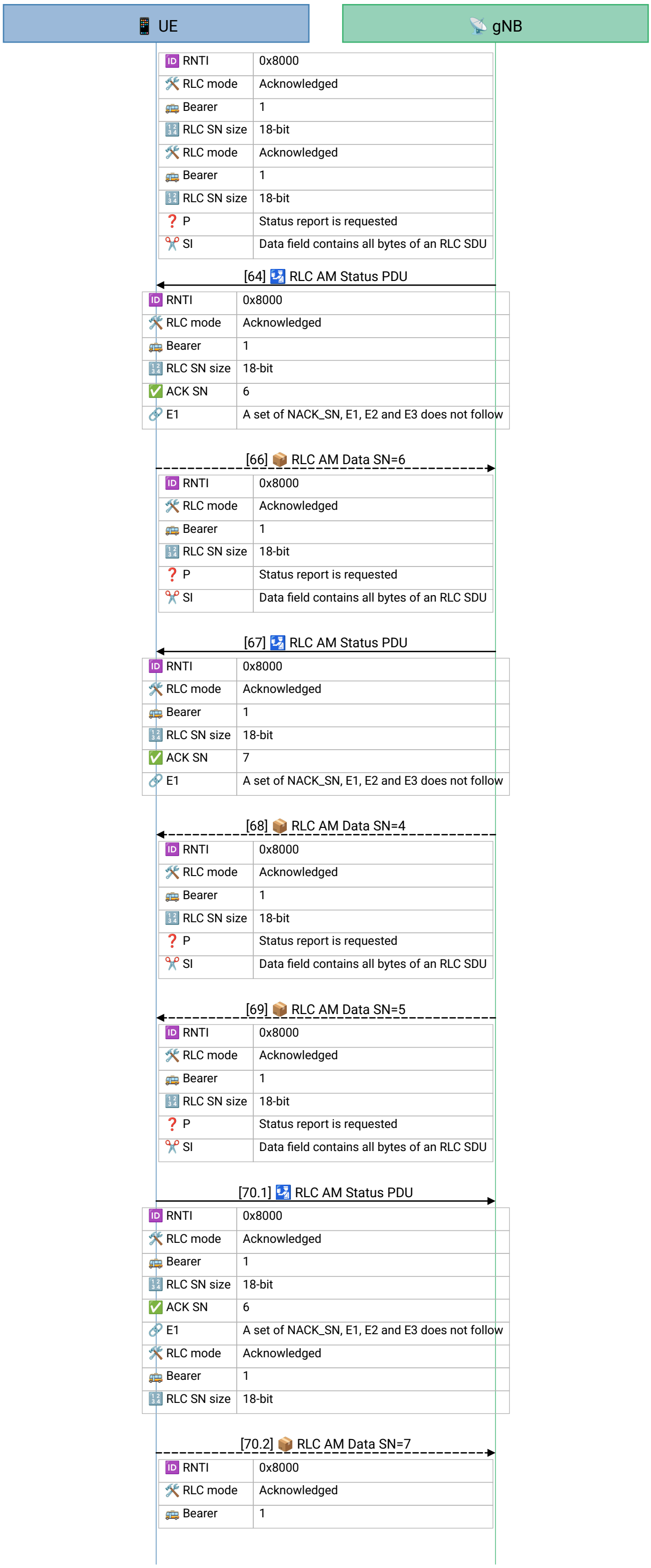
[Frame 55.2] Downlink SN=2, poll set, piggybacked behind that status in the same transport block.

[Frame 59] ACK_SN=3 — downlink SN 0 through 2 delivered.

[Frame 62] Downlink SN=3, poll set — a 791-byte SDU that fits its grant whole, so SI stays at "all bytes" and no segmentation is needed.

[Frame 63.1] ACK_SN=4 confirms the 791-byte SDU of frame 62.

[Frame 63.2] Uplink SN=5, poll set — piggybacked, and polled because the buffer is empty behind it.



[Frame 64] ACK_SN=6 — uplink SN 5 delivered.

[Frame 66] Uplink SN=6, poll set.

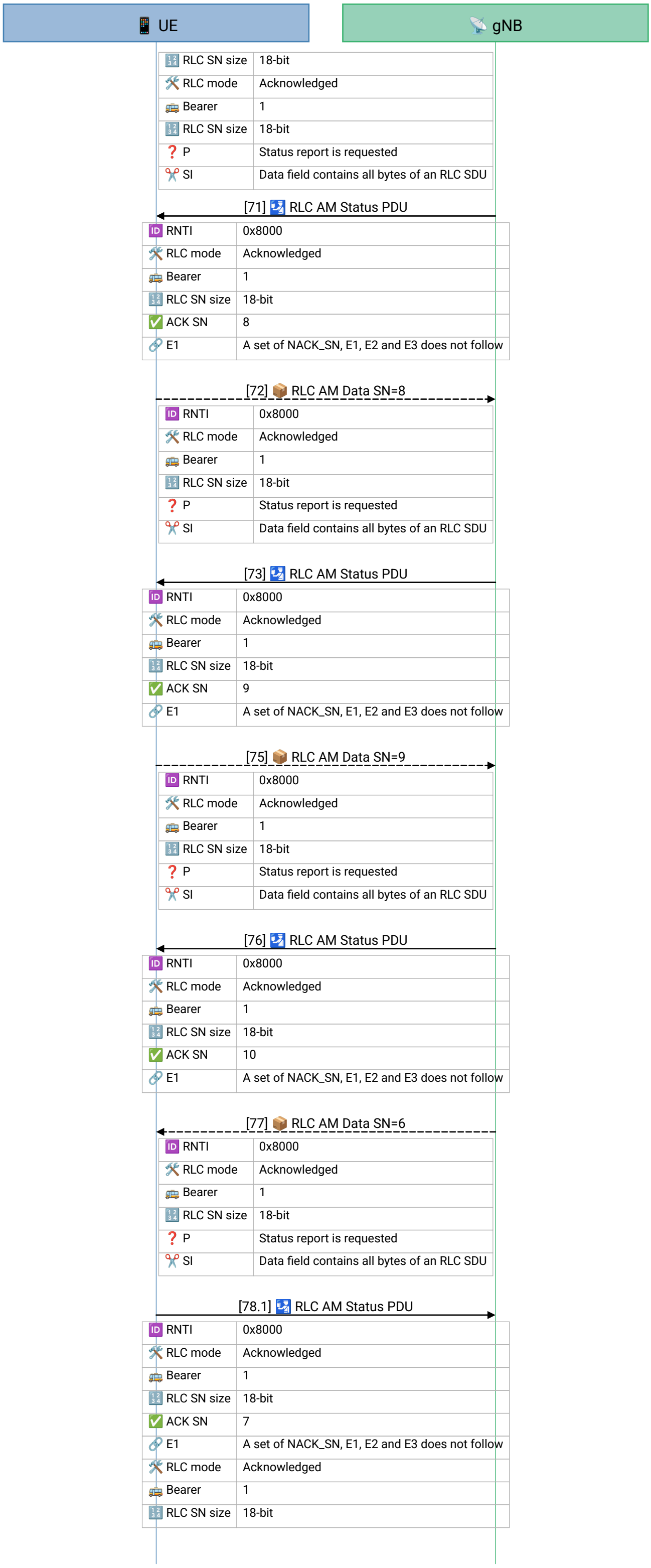
[Frame 67] ACK_SN=7 — uplink SN 6 delivered.

[Frame 68] Downlink SN=4, poll set.

[Frame 69] Downlink SN=5, poll set — one millisecond after frame 68. Two polls now sit unanswered, because the UE cannot reply until it is granted uplink airtime.

[Frame 70.1] Two polls, one answer. ACK_SN=6 covers downlink SN 0 through 5, so the polls of frames 68 *and* 69 are both satisfied by this single report. A STATUS PDU always describes the receiver's complete state up to ACK_SN, so a backlog of polls never means a backlog of statuses — which is exactly why RLC's control overhead stays flat as throughput rises.

[Frame 70.2] Uplink SN=7, poll set, sharing the transport block with the status above.



[Frame 71] ACK_SN=8 — uplink SN 7 delivered.

[Frame 72] Uplink SN=8, poll set — a 108-byte SDU.

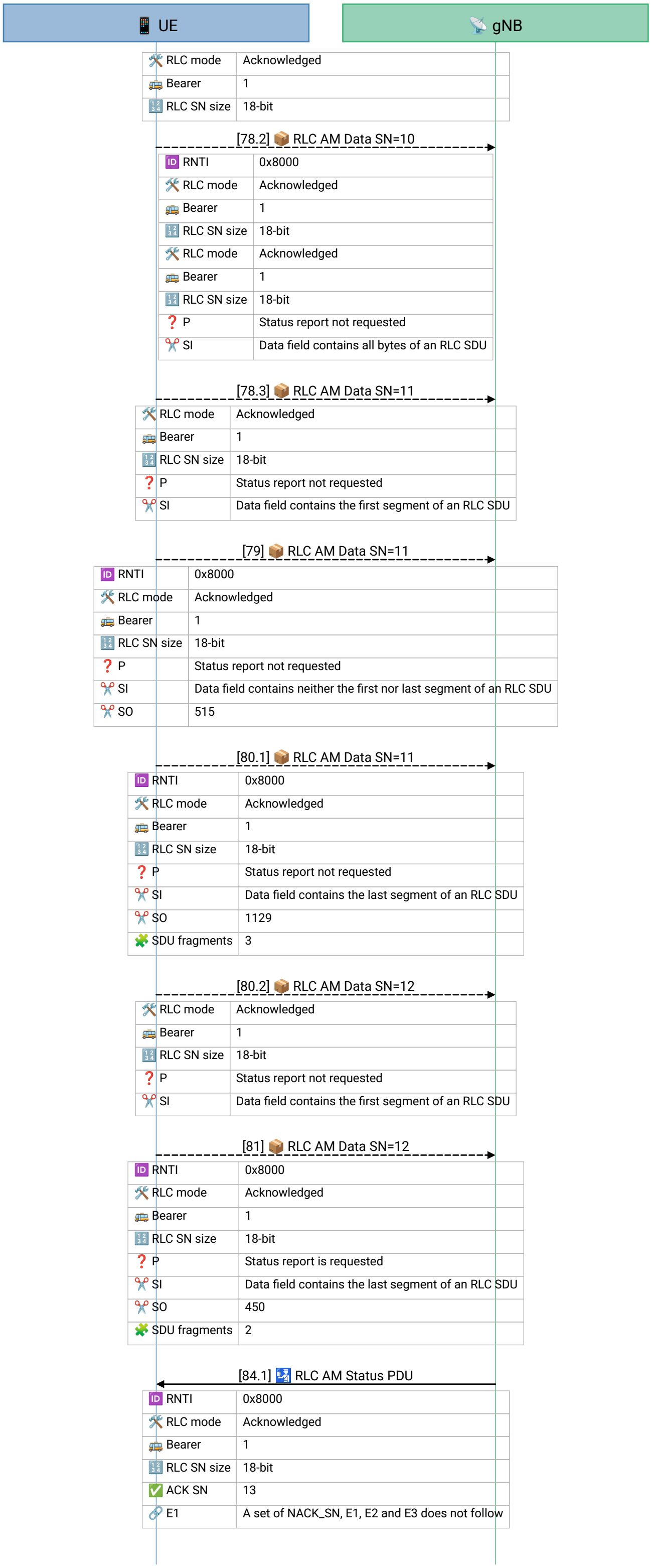
[Frame 73] ACK_SN=9 — uplink SN 8 delivered.

[Frame 75] Uplink SN=9, poll set, after a one-second lull. The sequence numbers pick up exactly where they left off: an idle AM entity keeps its state.

[Frame 76] ACK_SN=10 — uplink SN 9 delivered. What follows is the largest uplink burst on this bearer.

[Frame 77] Downlink SN=6, poll set.

[Frame 78.1] ACK_SN=7 clears downlink SN 6 and answers frame 77's poll, riding the same grant as the two data PDUs below.



[Frame 78.2] Uplink `SN=10`, a complete 88-byte SDU, no poll — the burst is starting.

[Frame 78.3] Uplink `SN=11`, `SI` = first segment, 515 bytes of a 1288-byte SDU. Compare with frame 50: the same three-PDU shape, produced by the same rule — fill the grant, split whatever does not fit, poll only at the end.

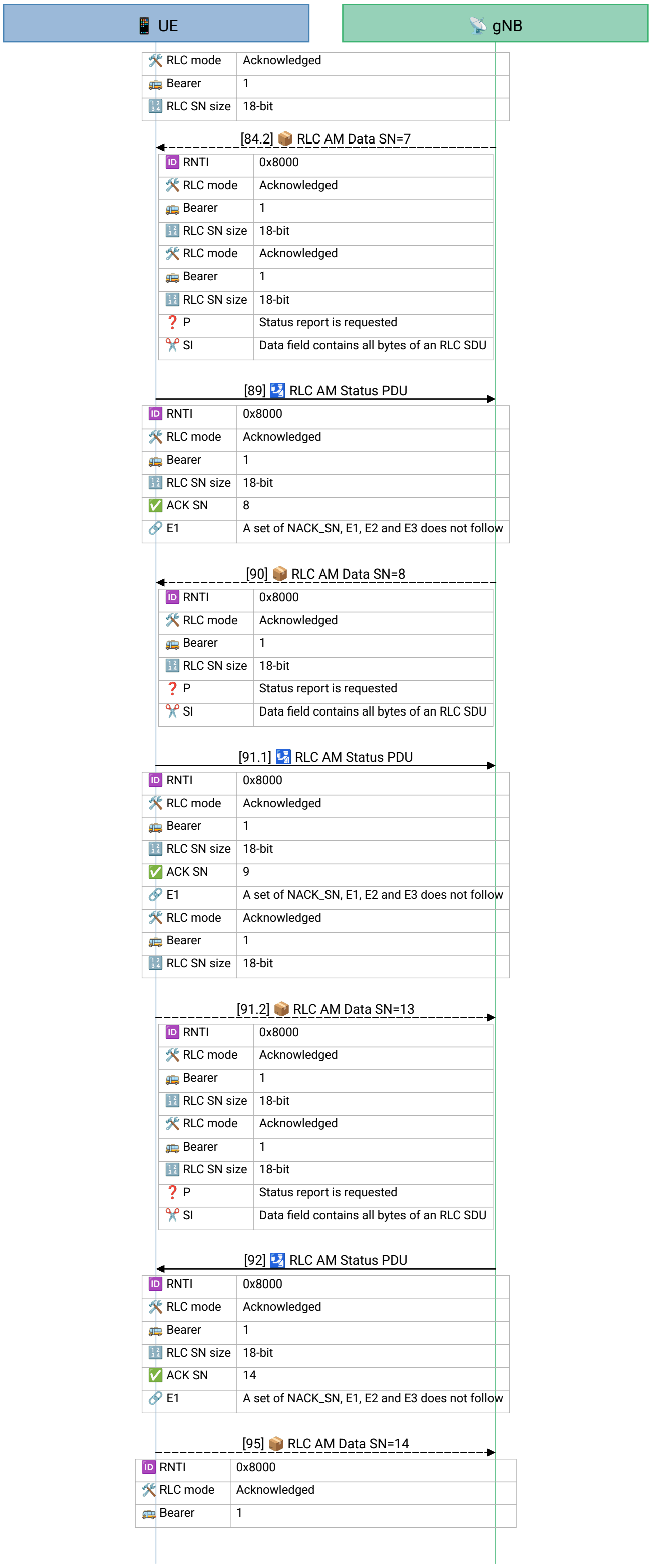
[Frame 79] `SN=11` middle segment, `SO=515`, 614 bytes of payload. No poll.

[Frame 80.1] `SN=11` last segment, `SO=1129`, which is 515 + 614. Reassembly: 3 fragments, 1288 bytes, as 515 + 614 + 159.

[Frame 80.2] Uplink `SN=12` opens immediately in the same transport block: `SI` = first segment, 450 bytes of a 988-byte SDU. Still no poll.

[Frame 81] `SN=12` last segment, `SO=450`, 538 bytes — a 2-fragment, 988-byte reassembly. The poll finally goes up. From frame 78 to here, six transmissions carried three SDUs, and exactly one of them asked for a report.

[Frame 84.1] Thirteen SDUs acknowledged in three bytes. `ACK_SN=13`: everything below SN 13 is delivered, which sweeps up the whole frame 78 to 81 burst, arriving 2.8 ms after the poll. Cumulative acknowledgment is the reason RLC AM's control overhead does not grow with the data rate.



[Frame 84.2] Downlink SN=7, poll set, piggybacked behind that status.

[Frame 89] ACK_SN=8 — downlink SN 7 delivered.

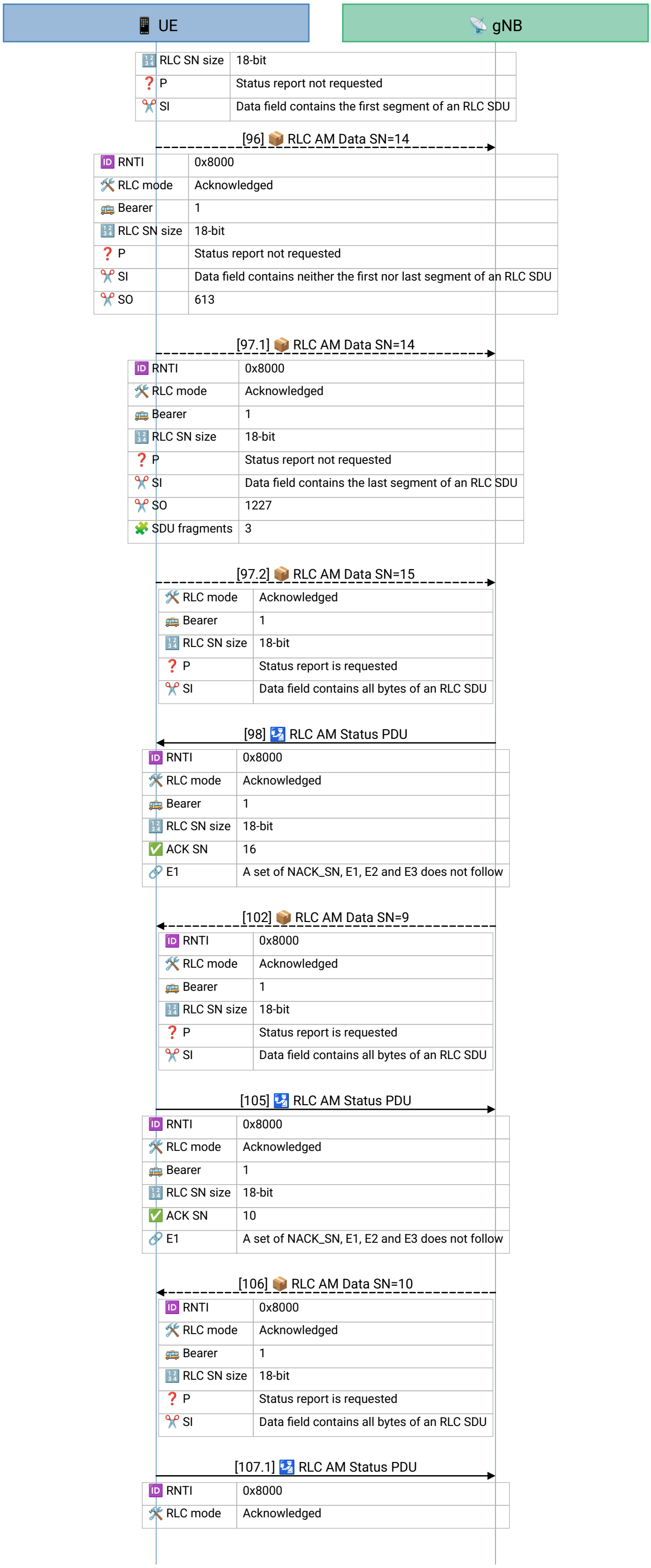
[Frame 90] A 1226-byte PDU, unsegmented. Downlink SN=8 with SI = "all bytes": the grant was large enough to take the whole 1223-byte SDU, so RLC did not split it. Hold this one next to frame 109, where a slightly larger SDU meets a slightly smaller grant and gets cut in two — the only difference between the two cases is how much room the scheduler happened to offer.

[Frame 91.1] ACK_SN=9 confirms the large downlink SDU of frame 90.

[Frame 91.2] Uplink SN=13, poll set, piggybacked.

[Frame 92] ACK_SN=14 — uplink SN 13 delivered.

[Frame 95] Uplink SN=14, SI = first segment, 613 bytes of a 1288-byte SDU, no poll. A third segmentation run begins — this one spread across three separate transport blocks rather than packed into shared ones, which changes nothing about how RLC describes it.



[Frame 96] SN=14 middle segment, SO=613，614 bytes.

[Frame 97.1] SN=14 last segment, SO=1227，which is 613 + 614. Reassembly: 3 fragments, 1288 bytes, as 613 + 614 + 61. No poll yet — one more SDU is queued.

[Frame 97.2] Uplink SN=15，a complete SDU, and **this** is where the poll lands. The rule has now held for every burst on this bearer: P is set on the last PDU RLC hands to MAC before its buffer empties, wherever in a segmentation run that happens to fall.

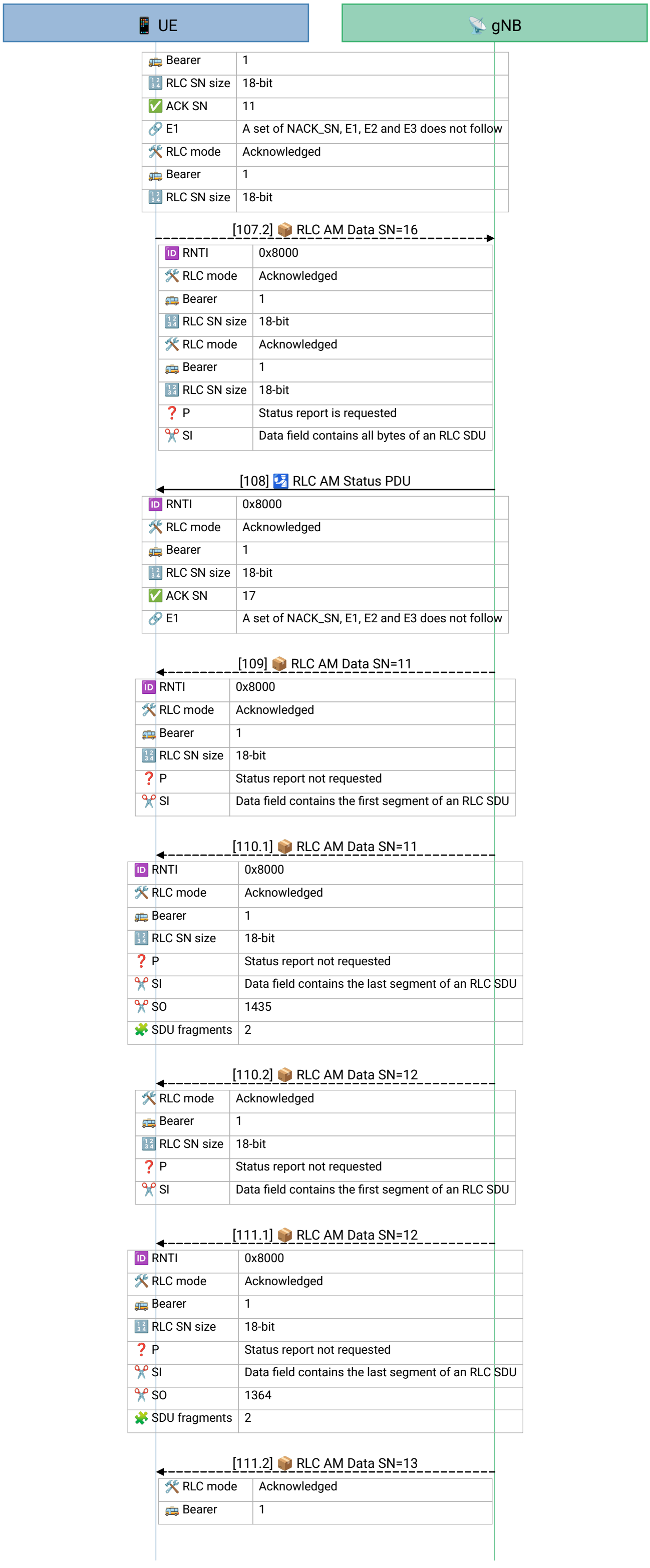
[Frame 98] ACK_SN=16 — uplink SN 0 through 15 delivered, closing the frame 95 to 97 run.

[Frame 102] Downlink SN=9，poll set.

[Frame 105] ACK_SN=10 — downlink SN 9 delivered, 18.3 ms after the poll. The uplink status latency is still an order of magnitude above the downlink's; the scheduler sets that pace, not RLC.

[Frame 106] Downlink SN=10，poll set — a 587-byte SDU.

[Frame 107.1] ACK_SN=11 — downlink SN 10 delivered.



[Frame 107.2] Uplink SN=16 , poll set, piggybacked.

[Frame 108] ACK_SN=17 — uplink SN 16 delivered. The downlink is about to show that it segments too.

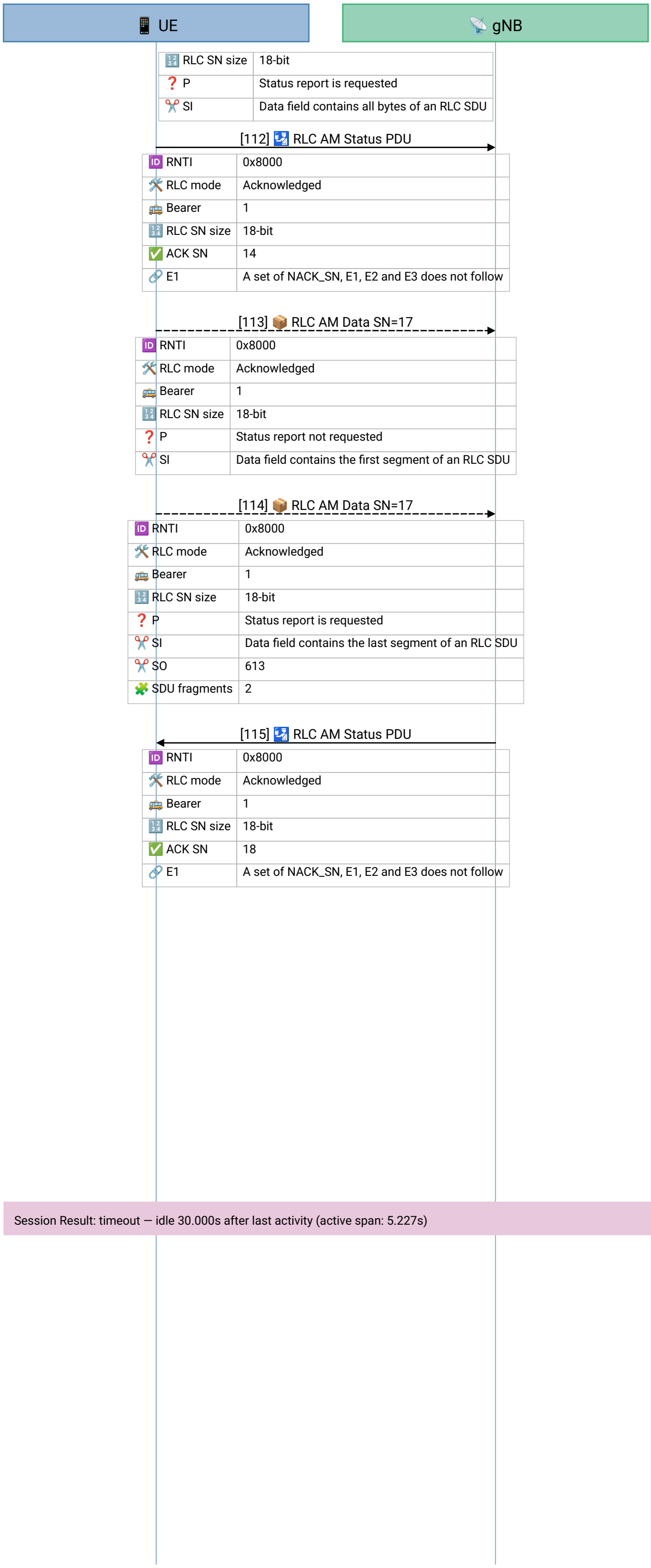
[Frame 109] The gNB segments. Downlink SN=11 , SI = first segment, no poll: a 1499-byte SDU meets a grant with room for a 1438-byte RLC PDU, so 1435 bytes go now and 64 bytes wait. Everything the uplink runs taught applies unchanged here — segmentation is symmetric, because both ends run the same AM logic against whatever their scheduler offers. Frame 90 sent a 1223-byte SDU whole; this one is 276 bytes larger and gets cut. That is the entire difference.

[Frame 110.1] SN=11 last segment, SO=1435 , the remaining 64 bytes — a 2-fragment, 1499-byte reassembly.

[Frame 110.2] Downlink SN=12 starts in the same transport block: SI = first segment, 1364 bytes of another 1499-byte SDU.

[Frame 111.1] SN=12 last segment, SO=1364 , 135 bytes — 2 fragments, 1499 bytes.

[Frame 111.2] Downlink SN=13 , a complete SDU, poll set. Same discipline as the uplink: the poll rides the last PDU of the burst, never the ones inside it.



[Frame 112] ACK_SN=14 — downlink SN 0 through 13 delivered, closing the two-SDU segmentation run of frames 109 to 111 with a single status.

[Frame 113] Uplink SN=17, SI = first segment, 613 bytes of a 984-byte SDU, no poll. The last segmentation run on this bearer.

[Frame 114] SN=17 last segment, SO=613, 371 bytes — 2 fragments, 984 bytes — with the poll set on the final piece.

[Frame 115] The bearer's closing acknowledgment. ACK_SN=18, answering 0.8 ms after the poll: uplink SN 0 through 17 — every SDU the UE sent on this bearer since frame 41, including six multi-segment SDUs — is confirmed delivered, and E1=0 says not one of them ever had to be reported missing.

The headline numbers for DRB 1: **18 uplink SDUs across 27 data PDUs, 14 downlink SDUs, 25 polls answered by 24 status reports, zero NACKs — and zero retransmissions.** The polls outnumber the statuses for the reason frame 70 showed: one status report can answer several outstanding polls at once.

The zero is not luck, and it is measurable: every poll on this bearer was answered. The slowest answer took 20.2 ms — but that one belongs to a *downlink* poll, frame 55's, answered by the UE at frame 59, so the 80 ms t-PollRetransmit is not the timer it was racing. Frame 38 signals that value inside the ul-AM-RLC block, where it governs the UE's transmitting entity; the uplink polls it does govern close far sooner, with frames 53 and 81 both answered 2.8 ms later. Nothing ever timed out in either direction, so nothing was ever re-sent.

Acknowledged Mode did its job so quietly here that the only evidence it was working is the absence of gaps. To see it actually intervene, look at the signaling bearer's diagram, where two polls go unanswered and the poll timer fires.

Session Result: timeout — idle 30.000s after last activity (active span: 5.227s)