[Frame 4] The bearer opens. This is the first RLC PDU (protocol data unit) of the capture, and it sets up everything that follows. Radio Link Control in Acknowledged Mode (AM) is the retransmission layer of the 5G radio stack: it numbers what it sends, chops it to fit whatever the scheduler grants, and refuses to let anything go missing.

Read the header fields left to right:

- `SN=0` — the first sequence number on this transmitting entity. RLC counts *SDUs* — service data units, the PDCP packets handed down to it — not bytes and not transmissions.
- `SI` = "all bytes of an RLC SDU" (the 2-bit Segmentation Info field, value 0) — nothing was split; one PDCP packet, one PDU.
- `P` = "status report is requested" — the poll bit. It obliges the peer to send a STATUS PDU back.

This entity is **SRB 1,** the signaling radio bearer: the packet trace shows bearer type SRB, bearer id 1, logical-channel ID 1, and a 12-bit sequence number. The SDU is an RRCSetupComplete with an initial NAS Registration Request inside. The PDU is 108 bytes — a 2-byte AM header plus a 106-byte SDU.

[Frame 5] The first acknowledgment, 0.8 ms later. A STATUS PDU is RLC's only control message, and it is tiny — 3 bytes on the wire. `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 — that is, SN 0 from frame 4 — 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.

Worth fixing in your head now: **there are no NACKs anywhere in this capture.** Every STATUS PDU you are about to see is a pure cumulative acknowledgment.

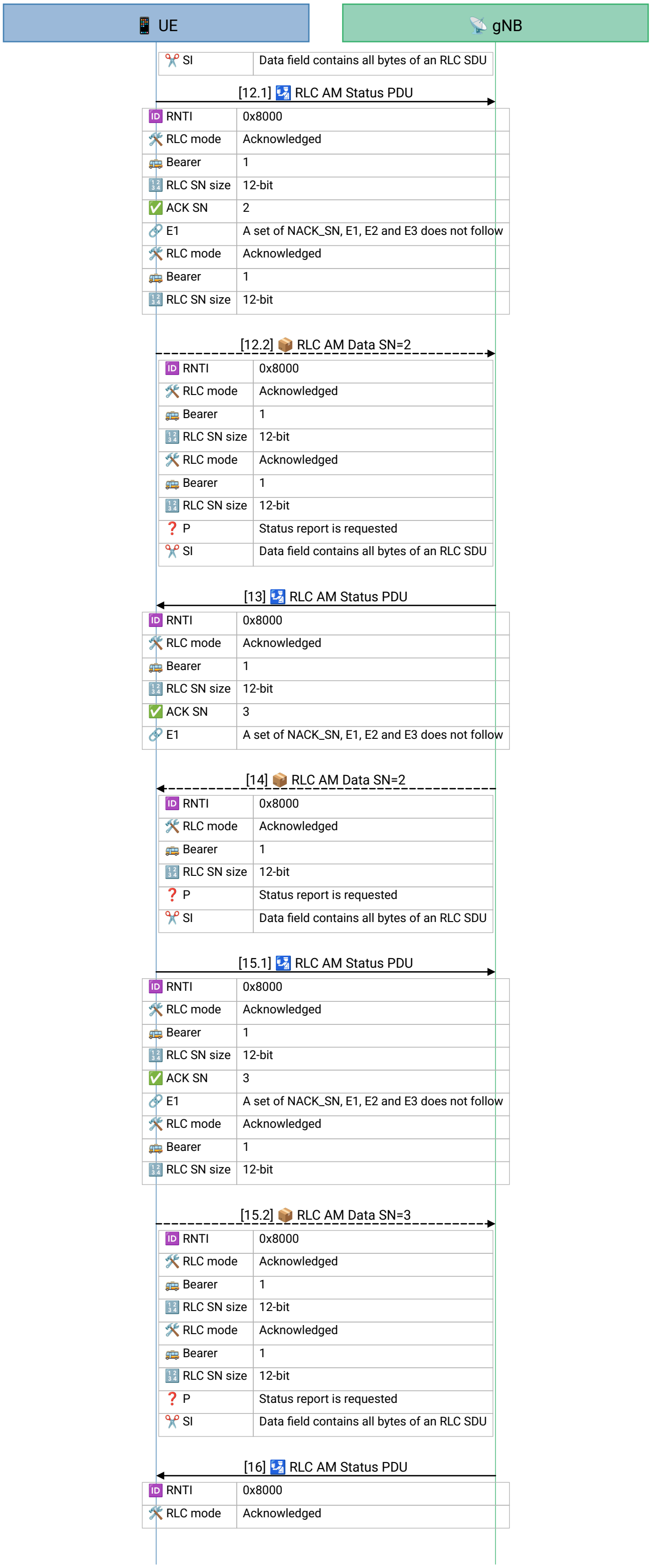
[Frame 7] The downlink has its own counter. The gNB now sends *its* SN 0. That is not a repeat of frame 4 — an AM bearer is two independent transmitting entities, one per direction, each with its own sequence-number space, its own poll timer, and its own retransmission buffer. This one carries an RRC DLInformationTransfer wrapping a NAS Authentication Request.

[Frame 8] Same acknowledgment, fourteen times slower. `ACK_SN=1` confirms the downlink SN 0 of frame 7 — but it arrives 11.2 ms after the poll, where the gNB answered frame 4 in 0.8 ms. The asymmetry is not RLC's doing: the gNB schedules its own downlink instantly, while the UE must be granted uplink airtime before it can transmit anything at all, even 3 bytes. The pattern holds for the whole session — every downlink status lands within 0.8 ms, every uplink status takes 10 to 19 ms.

[Frame 9] Uplink `SN=1`, `poll` set — the NAS Authentication Response. Signaling is bursty and arrives one message at a time, so on this bearer nearly every data PDU is also the last thing in the transmit buffer, which is precisely the condition that makes RLC set the poll bit (TS 38.322 §5.3.3.2).

[Frame 10] `ACK_SN=2`: uplink SN 0 and 1 are both delivered. The transmitter can now drop them from its retransmission buffer.

[Frame 11] Downlink `SN=1` — a NAS Security Mode Command inside a DLInformationTransfer. From here the NAS payloads are ciphered with keys this capture does not carry, so the inner detail goes dark; the RLC layer, which never sees plaintext anyway, carries on exactly the same.



[Frame 12.1] Two RLC PDUs, one radio transmission. This frame is a single MAC transport block carrying *both* a 3-byte STATUS PDU and an 85-byte data PDU on the same logical channel. The status half comes first: ACK_SN=2 clears the downlink SN 0 and 1.

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

[Frame 12.2] The data half of the same transport block: uplink SN=2 , poll set, an 83-byte SDU carrying a ciphered NAS message.

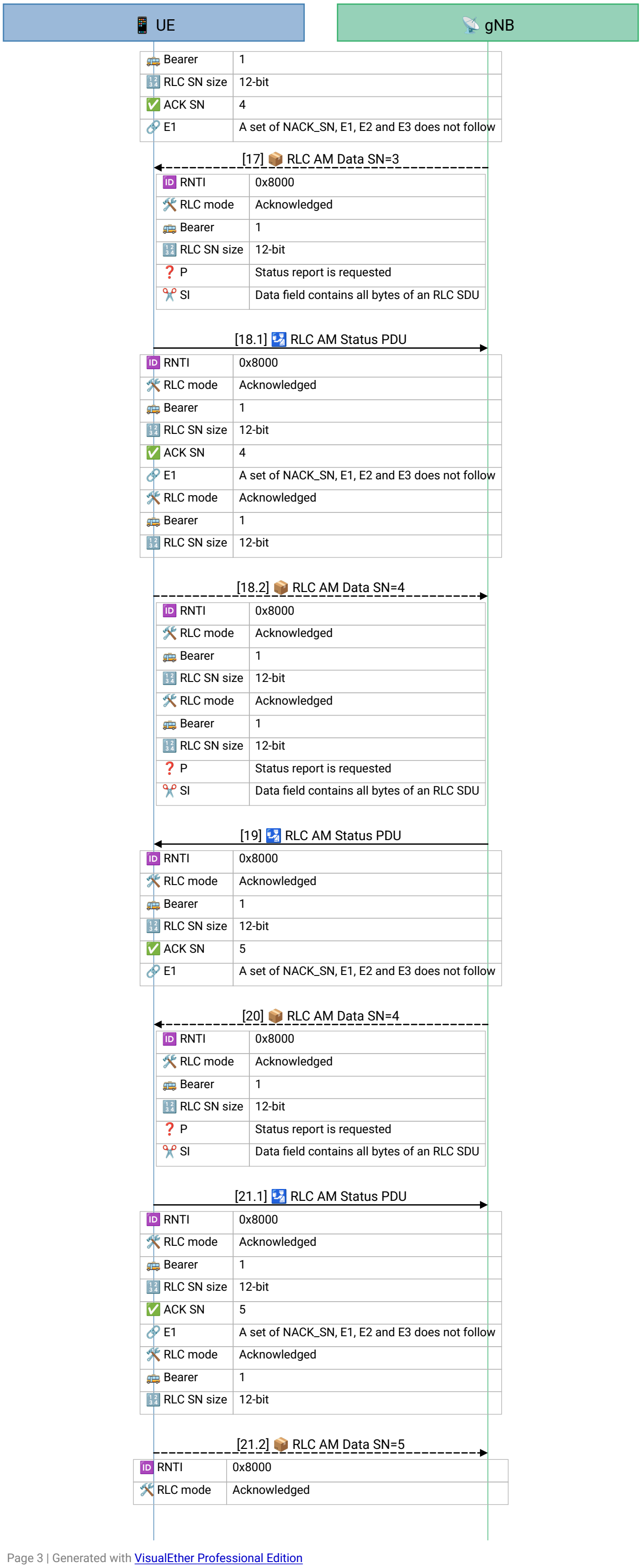
[Frame 13] ACK_SN=3 — uplink SN 0 through 2 acknowledged.

[Frame 14] Downlink SN=2 , another DLInformationTransfer with a NAS payload.

[Frame 15.1] ACK_SN=3 clears downlink SN 0 through 2, piggybacked ahead of the data PDU below, exactly as in frame 12.

[Frame 15.2] Uplink SN=3 , poll set — a 19-byte SDU.

[Frame 16] ACK_SN=4 — uplink SN 3 is in.



[Frame 17] Downlink SN=3 carries the RRC SecurityModeCommand. Access-stratum ciphering and integrity protection start after this exchange; note that RLC headers are never ciphered — the SN, `SI`, `SO` and poll bit stay in the clear so the peer's ARQ (automatic repeat request) machinery can work without keys. That is the whole reason this walkthrough is possible on a ciphered trace.

[Frame 18.1] `ACK_SN=4` acknowledges the SecurityModeCommand just received.

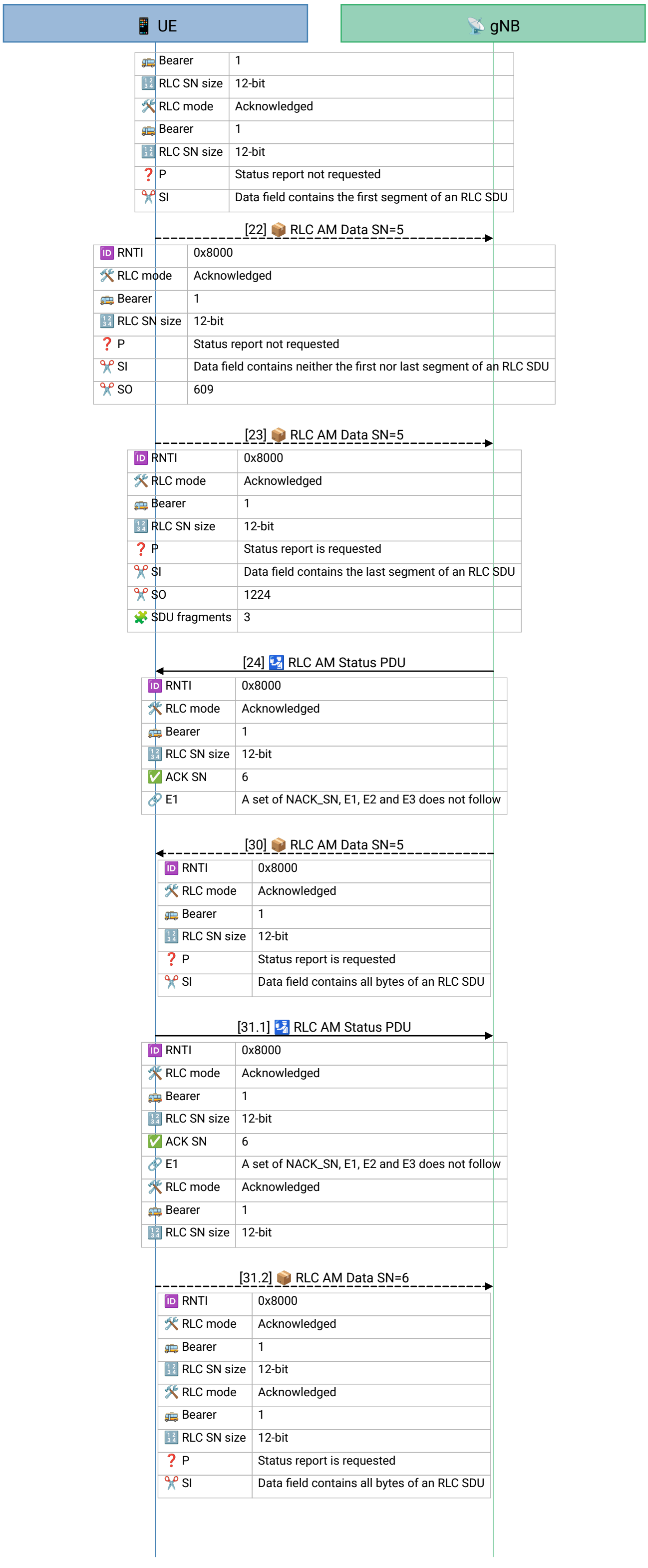
[Frame 18.2] Uplink `SN=4`, poll set — RRC SecurityModeComplete, an 8-byte SDU. Notice how small signaling PDUs are: an 8-byte SDU becomes a 10-byte RLC PDU inside a transport block with hundreds of bytes of padding.

[Frame 19] `ACK_SN=5`: uplink SN 0 through 4 delivered. Security is now up, and the next downlink message is the one that will stretch RLC.

[Frame 20] Downlink `SN=4` — RRC UECapabilityEnquiry. The answer to this is the largest signaling message on the bearer, and it is what forces the session's segmentation run.

[Frame 21.1] `ACK_SN=5` clears downlink SN 0 through 4 — the enquiry above is confirmed received. The UE now has a 1667-byte capability report to send, and the grant it just got is nowhere near big enough.

[Frame 21.2] Segmentation, part 1 of 3. `SI` = "first segment of an RLC SDU" (value 1), and P is **not** set. Both facts follow from one rule: RLC splits an SDU only because the MAC grant is smaller than the SDU, and it polls only when the buffer is about to run dry. Here 1058 bytes are still queued,



so no poll.

The PDU is 611 bytes: a 2-byte header (no SO field is present on a first segment, because the offset is zero by definition) plus 609 bytes of the SDU. Remember 609 — it is the offset the next segment will carry.

[Frame 22] Segmentation, part 2 of 3. `SI` = "neither the first nor last segment" (value 3) and `SO=609` — the Segment Offset, in bytes, from the start of the original SDU. That is exactly where frame 21 stopped. The header is now 4 bytes rather than 2, because a non-first segment must carry the 16-bit SO field; 619 bytes on the wire therefore means 615 bytes of payload, covering bytes 609 through 1223.

Still no poll: more of the SDU remains.

[Frame 23] Segmentation, part 3 of 3 — and the point of the whole exercise. `SI` = "last segment" (value 2), `SO=1224`, which is 609 + 615: the three segments tile the SDU without gap or overlap. Wireshark confirms the reassembly — 3 fragments, 1667 bytes, contributed as 609 + 615 + 443.

Two things to take away. First, **all three segments carry the same sequence number,** SN=5 — the number belongs to the SDU, and segmentation is described entirely by `SI` and `SO`. Second, the poll bit is set only *here,* on the last segment, because only now is the buffer empty. The receiver will therefore acknowledge the SDU once, not three times.

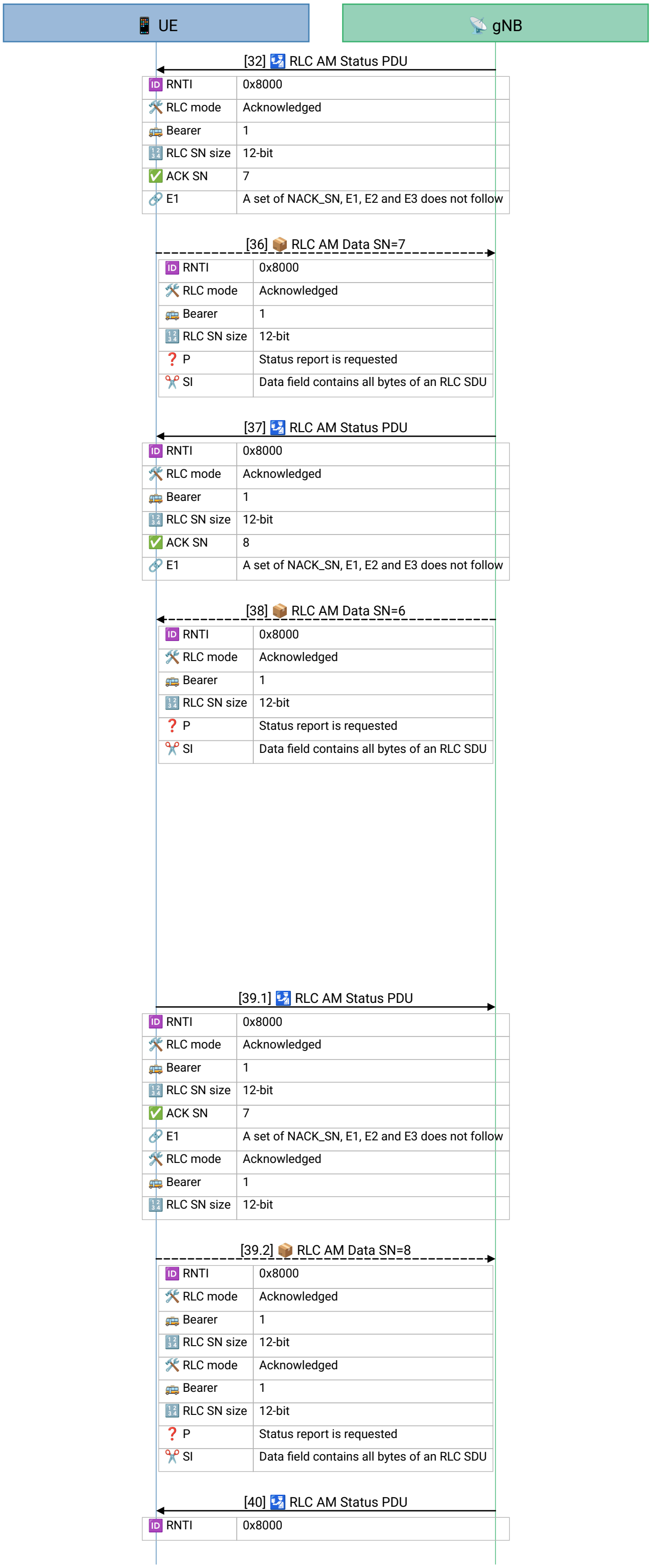
The reassembled SDU is the RRC UECapabilityInformation.

[Frame 24] One acknowledgment for three transmissions. `ACK_SN=6` — everything below SN 6 is delivered, and SN 5 counts as delivered only because all three of its segments arrived. This is the payoff of the previous three frames: ARQ state is per SDU, so a 1667-byte message costs exactly one entry in the retransmission buffer and one line in the status report, no matter how many pieces the scheduler forced it into.

[Frame 30] Downlink `SN=5`, poll set — a 62-byte SDU, back to ordinary one-PDU-per-message signaling.

[Frame 31.1] `ACK_SN=6` clears downlink SN 5 from frame 30.

[Frame 31.2] Uplink `SN=6`, poll set, piggybacked in the same transport block as the status above.



[Frame 32] ACK_SN=7 — uplink SN 6 delivered. A 1.37-second pause follows while the core network works; RLC simply idles, with nothing outstanding in either direction.

[Frame 36] Uplink SN=7 after the pause, poll set — a 101-byte SDU. The sequence numbers pick up exactly where they left off: an idle AM entity keeps its state.

[Frame 37] ACK_SN=8 — uplink SN 7 delivered.

[Frame 38] The most consequential SDU on this bearer. Downlink SN=6 is an RRC Reconfiguration, and decoded it adds a Data Radio Bearer (DRB) 1 on logical-channel ID 4, with its own RLC entity configured as:

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

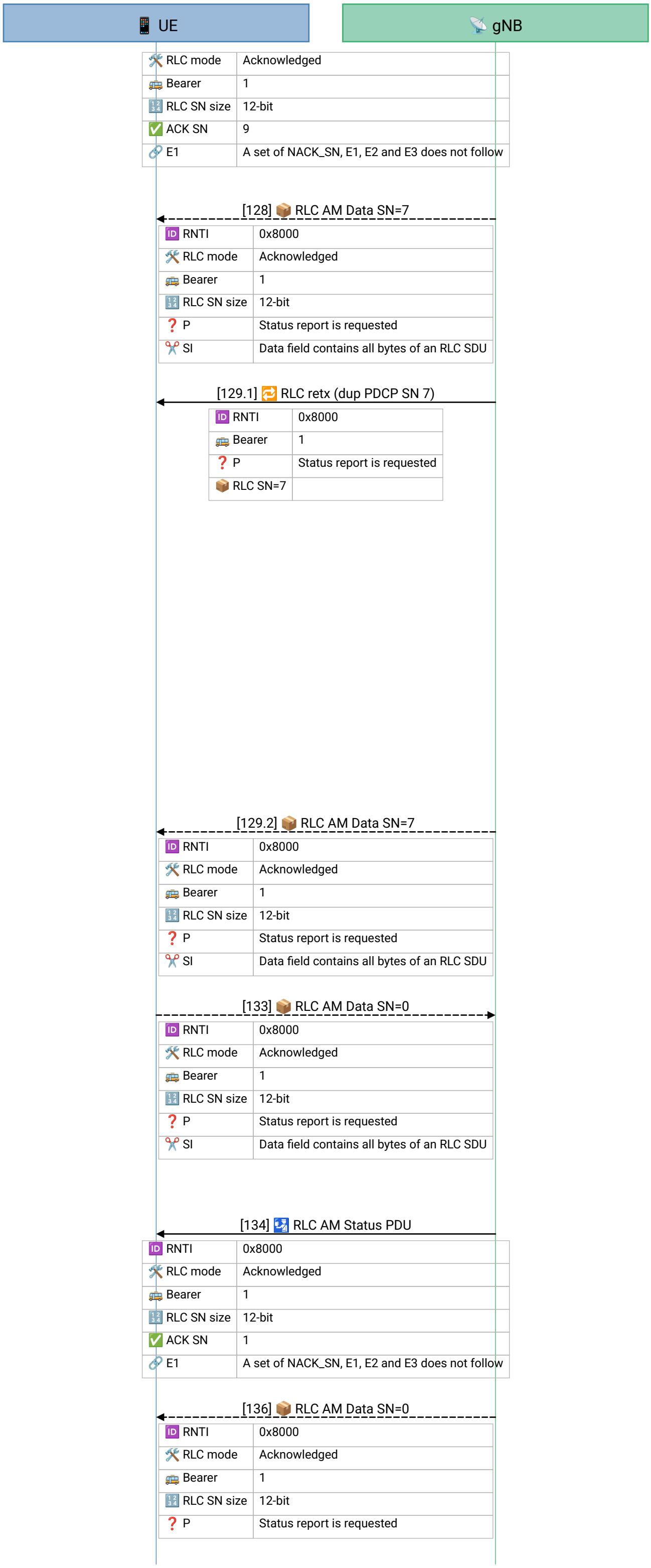
That bearer is a **separate RLC entity** with its own sequence-number space — nothing it sends shares a counter with anything on this diagram. Its story is the companion data-bearer walkthrough; this remark is where it begins.

The frame matters here for a second reason too. It is the only frame in the entire capture that signals any RLC configuration at all. **SRB 1 is never configured explicitly,** so this bearer runs the default acknowledged-mode profile of TS 38.331 §9.2.1 — 12-bit SN, `t-PollRetransmit` of 45 ms. That default becomes visible, and measurable, at frames 129 and 146.

[Frame 39.1] ACK_SN=7 acknowledges the reconfiguration of frame 38 at the RLC level — delivery confirmed, though the RRC-level confirmation is the PDU immediately below.

[Frame 39.2] Uplink SN=8, poll set — RRCReconfigurationComplete.

[Frame 40] The signaling bearer closes its books, then falls silent. ACK_SN=9 : uplink SN 0 through 8 — every SDU sent since frame 4, including the three-segment capability report



— is acknowledged, with no NACK ever raised. The retransmission buffer is empty.

What follows is a **10.3-second gap** on this diagram. The connection is not idle; the user-plane bearer that frame 38 created is carrying traffic throughout. The signaling bearer simply has nothing to say until the connection is torn down, and an AM entity with an empty buffer is completely silent — no keepalives, no periodic status, nothing.

[Frame 128] The bearer speaks again to end the connection. SN=7 continues directly from the SN 6 of frame 38, ten seconds earlier: this is still the same transmitting entity, and it kept its full state across the silence. The SDU is an RRC Release.

The poll is set, as on every data PDU of this bearer. This time nothing answers it — and that is what the next frame is about.

[Frame 129.1] The first retransmission — and it is a timer, not a NACK. PDCP flags the arriving SDU as a repeat of PDCP SN 7, which is the tell that RLC re-sent the PDU of frame 128 unchanged: same SN=7 , same 8-byte payload, poll set again.

Why it happened is provable from the trace rather than guessed:

- **No NACK caused it.** There is not a single `NACK_SN` in the capture; every status carries `E1=0`. **The interval is 44.9 ms,** measured from frame 128. `t-PollRetransmit` is the timer RLC starts when it sends a polling PDU; if no status arrives before it expires, the PDU goes back on the wire (TS 38.322 §5.3.3.4).
- **SRB 1 runs the TS 38.331 §9.2.1 default profile,** as frame 38 established, and that profile's `t-PollRetransmit` is ms45. The measured 44.9 ms is that timer firing.

For contrast, every *answered* poll on this bearer came back within 19.2 ms — comfortably inside the timer. Only the two polls that were never answered at all produced a retransmission.

The UE never does answer this one: it is already releasing the connection.

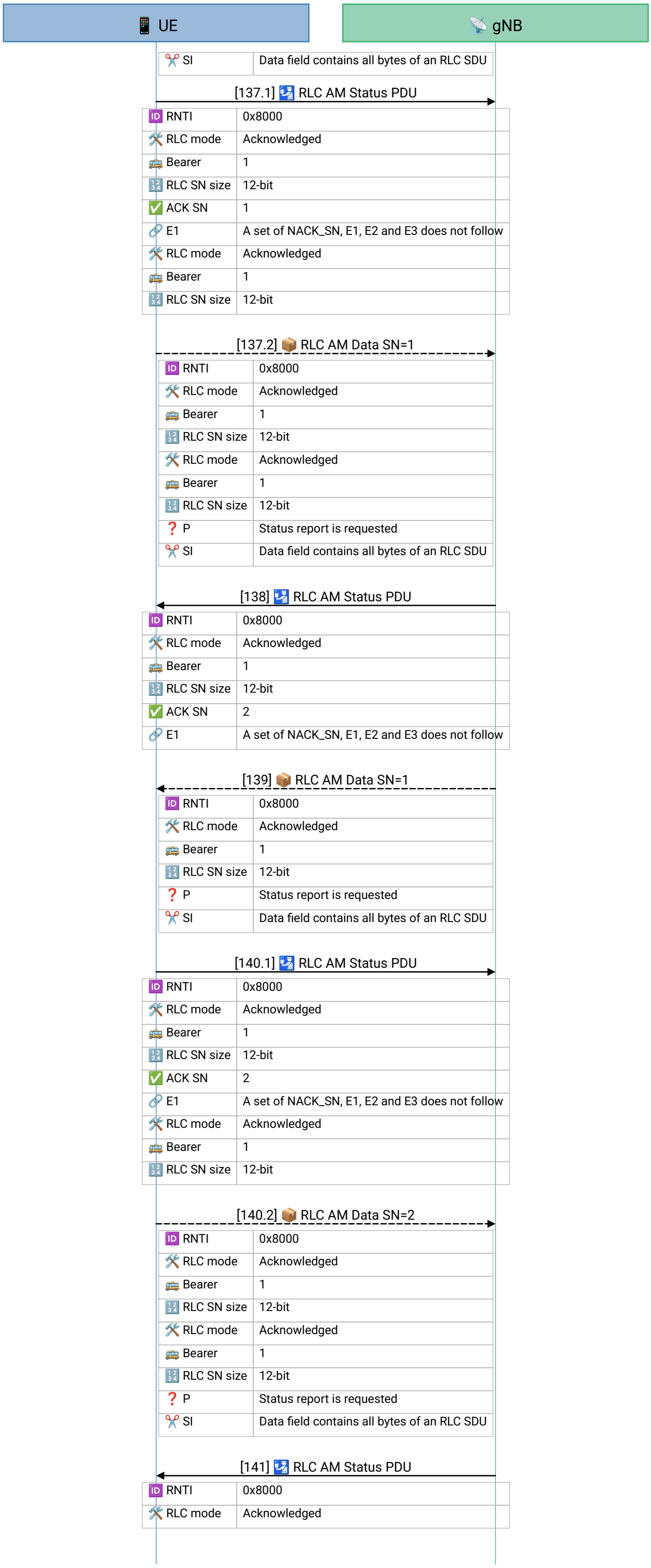
[Frame 129.2] The same PDU seen through the diagram's generic data-PDU template — **one transmission on the wire, two arrows.** Read it together with the retransmission arrow above: `SN=7` , `SI` = all bytes, poll set, identical to frame 128 in every field.

[Frame 133] A second connection, the same C-RNTI, a brand-new entity. `SN=0` on SRB 1 again — and this reset is not a reconfiguration but a fresh RRC connection: the packet trace shows a new UE context (UEId 2) reusing C-RNTI 0x8000 — the Cell Radio Network Temporary Identifier, the short handle the cell uses to address one connected UE — which is why this session, keyed on that identifier and the bearer, keeps it on the same lane.

An RLC entity is created and destroyed with the bearer, so its sequence numbers, transmit window and retransmission buffer all start from nothing. The SDU is an RRCSetupComplete carrying a NAS Service Request.

[Frame 134] `ACK_SN=1` — the new entity's first acknowledgment, 0.8 ms out, exactly as at frame 5. Everything this session taught over the previous 130 frames applies unchanged to a bearer that is one frame old.

[Frame 136] Downlink `SN=0` , poll set — the RRC SecurityModeCommand for the new connection. After this the RRC is ciphered with keys the capture does not carry, so the remaining SDU contents go opaque while every RLC header stays fully readable.



[Frame 137.1] ACK_SN=1 confirms downlink SN 0.

[Frame 137.2] Uplink SN=1 , poll set — RRC SecurityModeComplete, piggybacked in the same transport block.

[Frame 138] ACK_SN=2 — uplink SN 1 delivered.

[Frame 139] Downlink SN=1 , poll set. The payload is ciphered signaling from here on.

[Frame 140.1] ACK_SN=2 confirms downlink SN 1.

[Frame 140.2] Uplink SN=2 , poll set, piggybacked.

[Frame 141] ACK_SN=3 — uplink SN 0 through 2 delivered.

[Frame 142] Downlink SN=2, poll set.

[Frame 143] The last acknowledgment on this bearer. ACK_SN=3, E1=0: downlink SN 0 through 2 delivered, nothing missing. Every status report in the session has now been a clean cumulative ACK — 22 of them, not one NACK between them.

[Frame 145] Downlink SN=3, poll set — an 8-byte signaling SDU. The status that would clear it never arrives.

[Frame 146.1] The second retransmission closes the session — and confirms the first. PDCP flags a repeat of PDCP SN 3, and the interval from frame 145 is **44.9 ms:** the same figure, to within 21 microseconds, as the gap between frames 128 and 129. Two independent firings of the same default `t-PollRetransmit`, on two different RRC connections, with no NACK anywhere in between.

That is the honest summary of RLC ARQ on this bearer. Both retransmissions were driven by silence, not by a reported loss: the poll timer expired, so RLC re-sent. The radio link itself never dropped a single PDU.

[Frame 146.2] The generic view of that same single transmission — SN=3, SI = all bytes, P set. The session ends here, mid-ARQ, with one PDU still outstanding and its retransmission timer already reloaded. Nothing further arrives on this connection: the next frame on the air is a fresh random-access 32 seconds later, opening a third connection.

Session Result: timeout — idle 30.000s after last activity (active span: 16.715s)