Supplementary Material for LAMP: A Selective-ACK Transport Protocol for Reliable Bulk Data Transmission over LoRa Networks

Alessandro Monticelli, *Student Member, IEEE*

***Abstract*—This supplementary document provides extended implementation details, architectural state machines, and secondary parametric simulation blocks supporting the main manuscript "LAMP: A Selective-ACK Transport Protocol for Reliable Bulk Data Transmission over LoRa Networks". Specifically, we document: (i) the preliminary commercial transceiver proof-of-concept and its architectural lessons; (ii) the detailed Hardware Abstraction Layer (HAL) architecture on ESP32-S3 and desktop Unity test harness; (iii) formal finite state machine specifications for session lifecycle and data transfer; and (iv) extended ns-3 simulation studies evaluating message size scaling up to 62.2 kB (Block F), asymmetric acknowledgment loss sensitivity (Block G), retransmission poll rule ablation (Block H), and additional propagation channel profiles (Rural, Water, Static).**

***Index Terms*—LoRa, SACK, transport protocol, embedded systems, HAL, state machines, sensitivity analysis.**

I. Preliminary Proof of Concept: Commercial UART Transceivers

Prior to developing the native C++17 LAMP stack on Semtech SX1262 transceivers, an initial proof-of-concept was developed and evaluated using commercial Ebyte E220-series LoRa modules. These modules encapsulate physical-layer modulation behind an internal microcontroller running proprietary AT-command firmware accessible over an asynchronous UART interface.

While initial lab tests confirmed the basic feasibility of transport fragmentation over LoRa physical links, the trial exposed three critical architectural bottlenecks that motivated the ground-up redesign of LAMP:

1) **Opaque PHY and Inflexible MAC Control**: The proprietary internal firmware prevented direct access to raw baseband parameters, fine-grained symbol timing, carrier-activity detection (CAD), and exact FIFO interrupt timestamps, making deterministic timeout scheduling and half-duplex turnaround guard times impossible to guarantee.
2) **Inflated Protocol Overhead**: The vendor-mandated framing structure introduced substantial packet encapsulation bloat, reducing the net application payload capacity per frame and significantly increasing channel time-on-air (ToA).
3) **Absence of Receiver Feedback and Repair**: The commercial module firmware offered no native selective acknowledgment or sliding-window ARQ mechanisms, forcing early prototype firmware to resort to crude unacknowledged bursts or inefficient Stop-and-Wait retries.

These empirical findings motivated the complete transition to the open-source Semtech SX126x architecture driven via RadioLib and decoupled by a clean Hardware Abstraction Layer (HAL).

A. Monticelli is with the Department of Computer Science and Engineering (DISI), University of Bologna, 40126 Bologna, Italy (e-mail: alessandr.monticell4@studio.unibo.it).

II. Hardware Abstraction Layer and Implementation Architecture

A. Centralized Configuration and Parameter Decoupling

To eliminate hardcoded constants and enable deterministic tuning across heterogeneous embedded targets, all protocol operational parameters are centralized within a compile-time configuration namespace (`LoRaMultiPacketConfig`):

- **Memory and Concurrency Bounds**: Governs maximum concurrent reassembly sessions (`MAX_CONCURRENT_SESSIONS` = 10, reducible on memory-constrained microcontrollers), physical FIFO buffer bounds (`PHY_BUFFER_SIZE` = 256 bytes, matching transceiver hardware FIFO limits), and completed sequence deduplication depth (`MAX_COMPLETED_HISTORY` = 16).
- **Timing and Retransmission Policies**: Configures the default retry ceiling ($R_{\max} = 5$), minimum SACK timeout floor ($T_{\min} = 1000$ ms), safety multiplier ($\alpha = 1.5$), and half-duplex switching guard ($\delta_{\text{sw}} = 25$ ms).
- **Session Lifecycle Thresholds**: Defines handshake initiation timeouts (`DEFAULT_SYN_TIMEOUT_MS` = 3000 ms), connection inactivity teardown periods (`DEFAULT_CONN_INACTIVITY_TIMEOUT_MS` = 15000 ms), and the reassembly inactivity limit, measured from the last chunk received: the larger of `PRUNE_TIMEOUT_MS` = 15000 ms and $(R_{\max} + 1)$ SACK timeouts.

B. Hardware Abstraction Layer (HAL) Interface

The core transport logic communicates exclusively through abstract interfaces (`IRadio` and `IPlatform`). On physical nodes, `EspHal` wraps ESP32-S3 hardware resources:

- **SPI Bus Access**: High-speed SPI transfers (up to 16 MHz) for register configuration and FIFO packet buffering to the Semtech SX1262.

- **GPIO Interrupt Management**: Hardware interrupt dispatch for NSS, RST, BUSY, and DIO1 (packet transmission and reception completion triggers).
- **Monotonic Timing**: High-resolution platform timers backed by `esp_timer_get_time()` providing microsecond-level precision decoupled from RTOS tick scheduling.

C. Host-Based Unit Testing Suite

Following a test-driven development (TDD) discipline, the protocol core is verified on a desktop host environment using the Unity test framework within PlatformIO (72 native unit tests), executing without hardware-in-the-loop dependencies:

1) *Modbus CRC-16 Verification*: Frame integrity calculation across variable-length headers and padded chunk payloads.
2) *Reassembly State Engine*: Dynamic bitmap updates, out-of-order chunk reassembly, and duplicate suppression.
3) *Deterministic Pruning*: Verification that abandoned or stale sessions are evicted when inactivity timers expire.
4) *Concurrency Keys*: Proving that multiple active senders with identical message IDs remain isolated through $(\texttt{srcAddr}, \texttt{messageId})$ dual indexing.
5) *Simultaneous Open & Tie-Breaking*: Confirming deterministic resolution when two nodes initiate concurrent 3-way handshakes.

III. Finite State Machine Formal Specifications

Fig. 1 details the formal finite state machines governing the connection lifecycle and data transfer engines within LAMP.

A. Protocol Sequence Interactions

Fig. 2 and Fig. 3 illustrate the detailed message interaction flows for connection establishment, rejection handling, and SACK-ARQ selective repair.

B. Algorithmic Specifications: Receiver Reassembly and Transmitter SACK-ARQ

Algorithm 1 details the receiver-side out-of-order reassembly, CRC verification, and SACK bitmap generation logic, while Algorithm 2 formalizes the transmitter-side segmentation, burst dispatch, and selective repair loop.

C. Adaptive SACK Timeout Parameters

Table I details representative timeout parameters across spreading factors (BW 125 kHz, CR 4/5).

IV. Extended Simulation Studies: Blocks F, G, and H

A. Block F: Message Size Scaling (10 B to 62.2 kB)

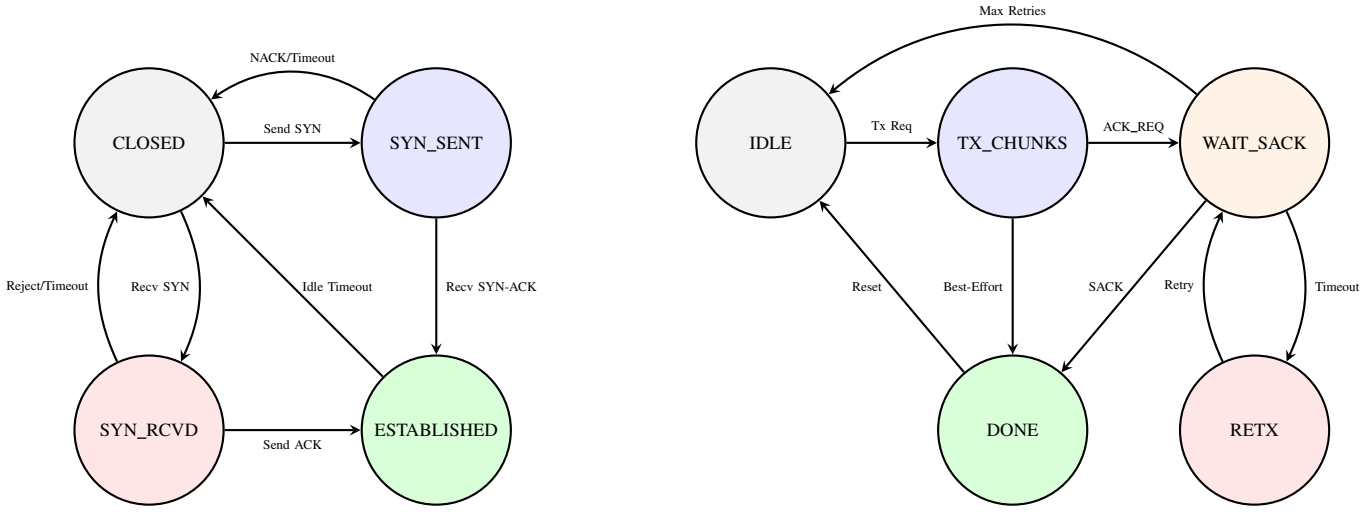
Block F evaluates protocol performance under large payload scaling at SF7 across message sizes $N \in \{20, 50, 100, 255\}$ chunks of 244 B (4.9 to 62.2 kB, reaching the maximum capacity of a single LAMP session). Comparative baselines include Stop-and-Wait, MPLR [1], and LoRaMesher [2].

Algorithm 1 Out-of-Order Reassembly and SACK Generation (Receiver)

Require: Received Frame F, Stride $S_{\max}$, Heap Margin M (16 kB)
Ensure: Reassembled Payload P or Drop Status
$(hdr, payload, crc) \leftarrow$ ParseFrame(F)
if ValidateCRC$(hdr, payload, crc) ==$ FAIL **or** $hdr.dstAddr \notin \{myAddr, \texttt{0xFF}\}$ **then**
  Drop(F); **return**
end if
if $hdr.dstAddr == \texttt{0xFF}$ **and** $(hdr.flags \ \& \ \texttt{FLAG_ACK_REQ})$ **then**
  Drop(F); **return**
end if
if $hdr.totalChunks == 0$ **or** $hdr.chunkIndex \geq hdr.totalChunks$ **then**
  Drop(F); **return**
end if
if IsCompleted$(hdr.srcAddr, hdr.msgId)$ **then**
  if $hdr.flags \ \& \ \texttt{FLAG_ACK_REQ}$ **then**
    SendTerminalSack$(hdr.srcAddr, hdr.msgId)$
  end if
  Drop(F); **return**
end if
$session \leftarrow$ FindSession$(hdr.srcAddr, hdr.msgId)$
if $session \neq$ NULL **and** $hdr.totalChunks \neq session.total$ **then**
  Drop(F); **return**
end if
if $session ==$ NULL **then**
  if FreeSlots$() == 0$ **or** AvailMem$() < hdr.totalChunks \cdot (S_{\text{pkt}} + S_{\text{node}} + S_{\max}) + M$ **then**
    Drop(F); **return** ▷ Sender exhausts $R_{\max}$
  end if
  $session \leftarrow$ AllocBuf$(hdr.totalChunks, S_{\max})$
end if
$idx \leftarrow hdr.chunkIndex$
if IsBitSet$(session.bitmap, idx)$ **then**
  SkipWrite(F) ▷ Duplicate chunk
else
  WriteChunk$(session.buffer, idx \cdot S_{\max}, payload)$
  SetBit$(session.bitmap, idx)$; $session.count \leftarrow session.count + 1$
end if
if $hdr.flags \ \& \ \texttt{FLAG_ACK_REQ}$ **then**
  SendSack$(hdr.srcAddr, hdr.msgId, session.bitmap)$
end if
if $session.count == session.total$ **then**
  $P \leftarrow$ ExtractPayload$(session.buffer)$
  DeliverToApp(P); MarkCompleted$(hdr.srcAddr, hdr.msgId)$; FreeSession$(session)$
end if

TABLE I: Adaptive SACK Timeout Values (BW 125 kHz, CR 4/5, 8-symbol preamble, explicit header, PHY CRC on; 255-byte data frame, 14-byte SACK for $N = 20$ chunks)

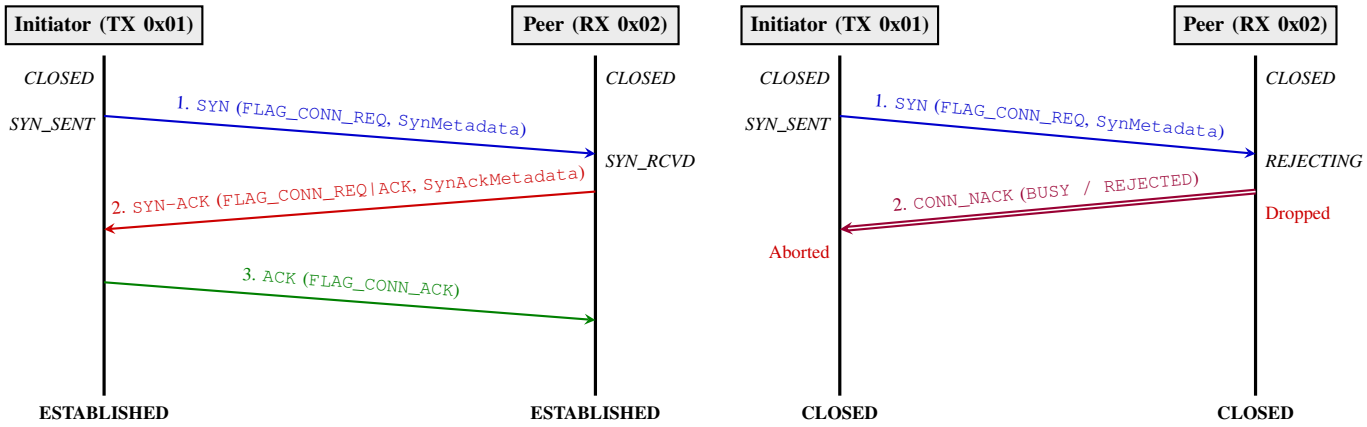
SF	$\widehat{T}_{\text{DATA}}$ (ms)	$\widehat{T}_{\text{SACK}}$ (ms)	$\alpha(\widehat{T}_{\text{DATA}} + \widehat{T}_{\text{SACK}}) + \delta$ (ms)	T_{timeout} (ms)
7	400	46	719	1000
8	707	82	1234	1234
9	1250	165	2173	2173
10	2296	289	3927	3927
11	5001	660	8541	8541
12	9019	1155	15312	15312



(a) Connection Session Lifecycle FSM (Stateful vs. Stateless)

(b) Data Transfer & SACK-ARQ FSM (Reliable vs. Unreliable)

Fig. 1: Finite State Machine (FSM) specifications for connection session lifecycle and data transfer modes.



(a) Successful 3-Way Handshake (3WHS) with Metadata Negotiation

(b) Connection Rejection (CONN_NACK) Failure Mode

Fig. 2: Sequence interaction flows for 3-Way Handshake (3WHS) connection setup with dynamic metadata negotiation and rejection handling.

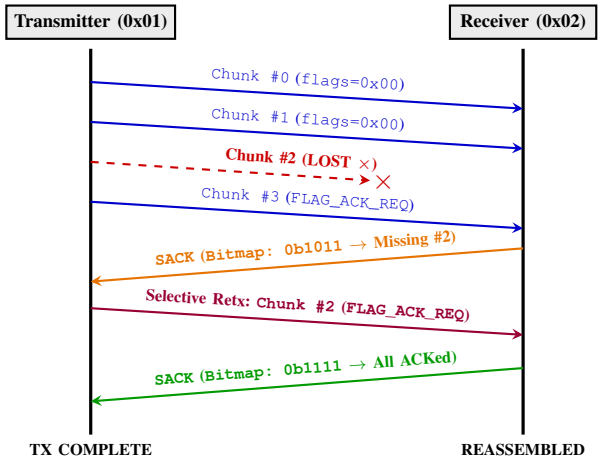


Fig. 3: End-to-end data streaming flow, frame loss detection, and SACK-driven selective retransmission interaction.

Fig. 4 depicts the 90% PDR delivery range in the Urban channel and goodput under static lossless conditions. The delivery range of LAMP remains invariant to message size (Mode 4 maintains 4.13 km; Mode 3 maintains 4.56 km up to 100 chunks and 4.13 km at 255 chunks). Conversely, architectures assigning individual per-frame retry budgets experience compounding degradation as N increases: Stop-and-Wait drops from 3.38 km to 2.26 km, while LoRaMesher ($R = 10$) falls from 4.13 km to 3.09 km.

For MPLR, behavior depends on the interpretation of its retry budget: if $R_{\max}$ applies globally across the entire multi-batch transfer, range collapses to 3.7 km at $N = 50$ and fails to reach 90% at $N = 255$. If $R_{\max}$ is refreshed per batch of 40 frames, MPLR maintains 4.13 km, matching Mode 4. Under this favorable reading, LAMP retains higher goodput due to its compact 9-byte header (+3% faster at 62.2 kB, delivering in 103 s vs. 106 s).

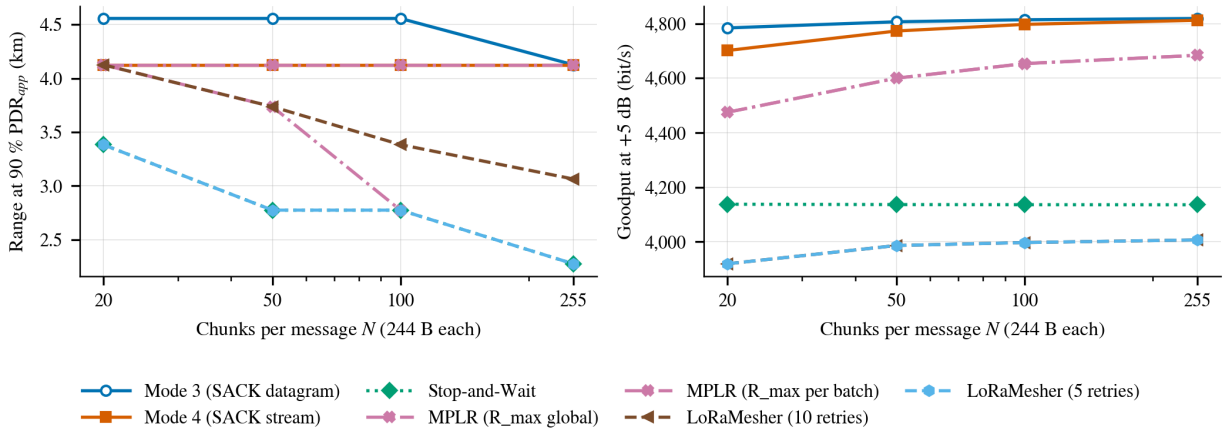


Fig. 4: Urban range (largest distance with mean $PDR_{\text{app}} \geq 90\%$) and goodput at +5 dB (static channel) versus the number of 244-byte chunks per message, SF7, 50 seeds. MPLR is evaluated under both global retry budget and per-batch refresh.

Algorithm 2 LAMP Payload Transmission and SACK-ARQ (Transmitter)

Require: Payload P (size $L > 0$), $dstAddr$, Chunk Size $S = S_{\max}$, Mode M
Ensure: Delivery Status (`OK` or `FAIL`)
$N \leftarrow \lceil L/S \rceil$
if $N > 255$ **then**
  return `ERR_PAYLOAD_TOO_LARGE`
end if
$msgId \leftarrow$ GetNextMsgId()
for $i \leftarrow 0$ **to** $N-1$ **do**
  $cLen \leftarrow (i == N-1)?(L - i \cdot S) : S$
  $flags \leftarrow (M \in \{\text{Mode 3}, \text{Mode 4}\}$ **and** $i == N-1)?$`FLAG_ACK_REQ : 0x00`
  $hdr \leftarrow \text{BuildHdr}(src, dst, msgId, N, i, cLen, flags)$
  $crc \leftarrow \text{CalcCRC16}(hdr, P[i \cdot S : i \cdot S + cLen])$
  $F_i \leftarrow \text{SerializeFrame}(hdr, P[i \cdot S : i \cdot S + cLen], crc)$
  Transmit(F_i)
end for
if $M \in \{\text{Mode 1}, \text{Mode 2}\}$ **then**
  return `OK` ▷ Unreliable delivery complete
end if
$retries \leftarrow 0$; $poll \leftarrow N-1$
while $retries \leq R_{\max}$ **do**
  $sack \leftarrow \text{WaitForSack}(T_{\text{timeout}})$
  if $sack ==$ TIMEOUT **then**
    if $retries == R_{\max}$ **then**
      break
    end if
    Transmit(F_{poll}) **with** `FLAG_ACK_REQ`; $retries \leftarrow retries + 1$
  else if IsAllAcked($sack.bitmap$) **then**
    return `OK` ▷ Full payload delivery confirmed
  else
    if $retries == R_{\max}$ **then**
      break
    end if
    $missing \leftarrow \text{MissingChunks}(sack.bitmap)$
    for $idx \in missing$ **do**
      $cLen \leftarrow (idx == N-1)?(L - idx \cdot S) : S$
      $flags \leftarrow (idx == \text{Last}(missing))?$`FLAG_ACK_REQ : 0x00`
      $hdr \leftarrow \text{BuildHdr}(src, dst, msgId, N, idx, cLen, flags)$
      $crc \leftarrow \text{CalcCRC16}(hdr, P[idx \cdot S : idx \cdot S + cLen])$
      $F_{idx} \leftarrow \text{SerializeFrame}(hdr, P[idx \cdot S : idx \cdot S + cLen], crc)$
      Transmit(F_{idx})
    end for
    $poll \leftarrow \text{Last}(missing)$; $retries \leftarrow retries + 1$
  end if
end while
return `ERR_MAX_RETRIES_EXCEEDED`

B. Block G: Asymmetric SACK Feedback Loss Robustness

Block G investigates performance when feedback frames (SACK, SYN-ACK, ACK) are subjected to loss probability $p_{\text{ack}} \in \{0, 20, 40\}\%$, concurrent with data frame loss $p \in [0, 50]\%$ (SF7, $N = 20$).

Because lost SACK feedback triggers the timer-driven poll mechanism, the penalty is predominantly manifested as latency rather than transfer aborts. At $p = 20\%$ and $p_{\text{ack}} = 20\%$, Mode 3 maintains 98% transfer completion, with elapsed time increasing moderately from 11.1 s to 11.7 s; at $p_{\text{ack}} = 40\%$, completion remains at 92% (13.1 s). In Mode 4, lost SYN-ACK frames incur handshake retries, increasing latency to 15.8 s at $p = 20\%, p_{\text{ack}} = 40\%$.

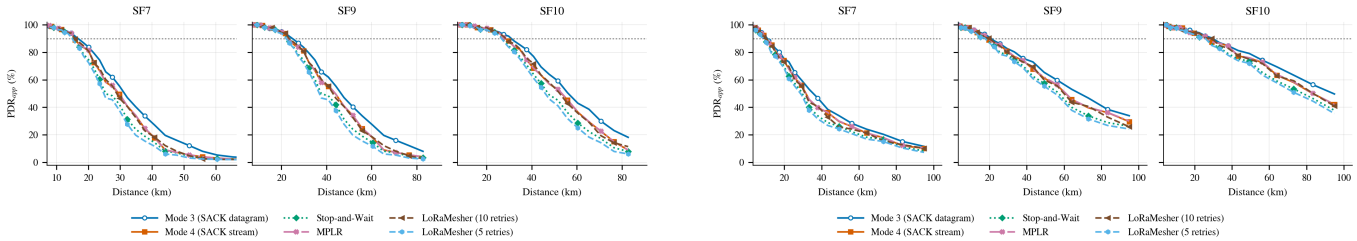
In contrast, Stop-and-Wait exhibits severe sensitivity to feedback losses: at $p = 30\%$, its completion rate plunges from 100% (lossless feedback) down to 84% at $p_{\text{ack}} = 20\%$ and 46% at $p_{\text{ack}} = 40\%$, whereas LAMP Mode 3 preserves 90% and 76% completion under the same harsh feedback loss regimes.

C. Block H: Retransmission Timeout and Poll Rule Ablation

Block H isolates the architectural benefit of the LAMP timeout poll rule (which retransmits the final chunk carrying `FLAG_ACK_REQ` upon SACK expiration, rather than restarting full-window retransmission). Without this targeted poll rule, Mode 3 completion drops drastically under high frame loss: at $p = 30\%$, completion falls from 92% to 80%; at $p = 40\%$, completion plummets from 56% to 40%. This confirms that targeted polling prevents catastrophic retry exhaustion.

D. Rural, Water, and Static Propagation Comparisons

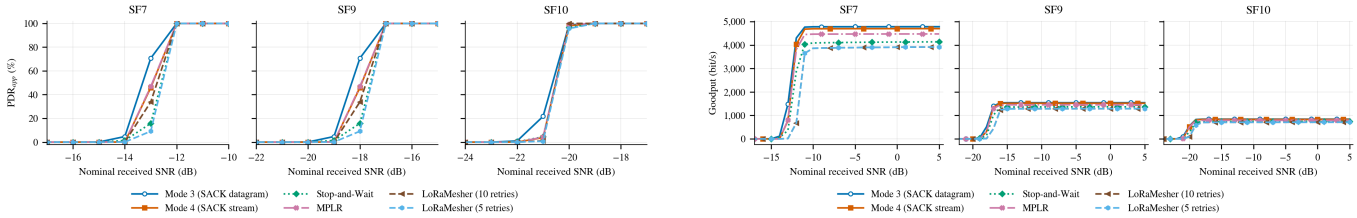
For completeness, Figs. 5a, 5b, and 6 provide the detailed comparative performance curves of LAMP against MPLR, LoRaMesher, and Stop-and-Wait across Rural, Water, and Static AWGN channel environments, complementing the Urban channel results reported in the main manuscript.



(a) Rural Propagation Channel

(b) Water Propagation Channel

Fig. 5: PDR_{app} versus distance across alternative propagation environments for LAMP and related protocols (SF7, 50 seeds, 90% PDR threshold marked with dotted line).



(a) PDR vs. Nominal SNR in Static Channel

(b) Goodput vs. Nominal SNR in Static Channel

Fig. 6: Performance comparison under Static AWGN conditions versus nominal SNR for LAMP and related protocols (SF7, 4.88 kB transfer, 50 seeds).

REFERENCES

[1] T. Chen, D. Eager, and D. Makaroff, "Efficient image transmission using LoRa technology in agricultural monitoring IoT systems," in *2019 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2019, pp. 937–944.

[2] J. M. Solé, R. Pueyo Centelles, F. Freitag, R. Meseguer, and R. Baig Viñas, "Large and reliable data transfer service for LoRa mesh network applications," *Computer Communications*, vol. 248, p. 108404, 2026.