



# Introduction

This document contains some of the work from performing downlink experiments from the TTI console to device, TBWL100\_10, to fix the Rx timing and then the channel mask problem.

I was ultimately unable to get the device to accept a downlink, so could not fix the Rx timing or the channel mask problem for device TBWL100\_10. Had this worked, I would have performed the same downlink exercise on the other 4 devices with poor PC rates.

## TLDR

**Background:** Longview is an IoT system in Starkville, MS that includes 7 Brownan LoRaWAN sensors and a single RAK7268C gateway. The IoT gateway was offline from Aug 2-11, 2025. Before the gateway outage, all devices maintained ~94% packet completion rate (PC Rate) – that is ~94% of packets sent by the device were received by the LoRaWAN Network Server.

**Problem Statement:** After gateway recovery, 2 devices returned to ~94% PC Rate but 5 devices dropped to ~34%.

**Root cause of the PC Rate performance decline after the gateway outage:** Post-outage **MAC state drift** (channel mask + RX timing) on five devices.

**Proposed Fix: RX timing first** (RX1=1s, RX2=923.3/DR8), verify **confirmed downlink ACK**, then **LinkADRReq** to the gateway's sub-band; repeat per device and monitor PC Rate recovery.

**Result:** Working on 1 of the 5 poor performance devices, I was unable to get the device to accept any downlinks, thus was unable to correct the MAC State drift. Since it is unlikely that any of the devices will initiate an OTAA rejoin on their own (based on previous experience working with the Brownan sensors), the only way to fix the PC Rate performance is to physically reset the devices by pulling the battery, shorting the battery terminals, then reinserting the battery.

## Table of Contents

### Contents

Introduction .....	1
TLDR .....	1
Table of Contents .....	1
Final Results .....	2
Rx Timing Conundrum.....	2
Problem Statement.....	3
Root cause (most likely).....	3
Rx Timing Mismatch Detailed Explanation .....	4
Fix strategy (remote-first) .....	4
Deeper Dive into the Sub-band Mismatch Concept .....	5
RAK7268C Configuration.....	7
Implementing the Rx Timing Fix .....	8
Plan.....	8



Executing the Fix .....	10
Before Change.....	10
Change .....	13
After Change .....	19
Implementing the push <b>LinkADRReq</b> to constrain the device to the <b>gateway's actual 8-channel block</b> . ....	22
Plan.....	22

## Final Results

After several days of trying to fix the Rx Timing (the first step of the 2-step recommended fix process), I haven't been able to get the device to ack the downlink. If I can't get a downlink to the device, I can't change the Sub Mask and thus won't be able to improve the 34% Packet Completion rate of the 5 devices without doing a physical reset of the device.

## Rx Timing Conundrum

Specifically, the device is still **not hearing** any of the confirmed downlinks, so nothing you queue (RX1 delay change, channel mask, DevStatusReq, etc.) is getting through.

### Why it keeps missing

- Every ns.down.data.schedule.\* and ns.down.transmission.success you've shared still shows **rx1\_delay: 5**.
- Your device is almost certainly listening at **RX1 = 1s** (and RX2 = 2s).
- So the NS is transmitting at **5s (and 6s)**, while the device only opens its windows at **1s (and 2s)** → it will **miss 100%** of downlinks.
- Because no downlink is ever "heard," the **RX1=1s fix cannot be applied** (catch-22).

### What this means (practically)

Changing "Desired Rx1 delay" in the console only **queues** an RXTimingSetupReq; it **doesn't take effect** until the device actually receives a downlink. Since it's deaf to 5s/6s frames, it never will.

### Your viable recovery paths

Pick one of these; they're the only ways to break the loop:

1. **Force a rejoin (best, fastest) — on-site battery pull**
  - Ask the local contact to **remove and reinsert the battery** on TBWL100\_10.
  - Watch for a new **JoinRequest / JoinAccept** in Live Data.
  - Immediately queue (in this order):
    1. **RX timing**: Desired Rx1 delay = **1 sec**, Rx2 = **923.3 / DR8**, Offset = **0** (Replace queue, Confirmed).
    2. **Tiny confirmed test downlink** (JSON AQ== on FPort 1).
    3. **FSB1 channel mask** via LinkADRReq (or use the Channels UI to enable **8–15 only**).
  - Verify ACK on next uplink and **LinkADRAbs (success)**.
2. **Gateway assist (if you can get someone to log in)**
  - Temporarily switch the RAK7268C to **listen across multiple sub-bands** (or "All uplink channels").
  - This won't fix timing by itself, but it **maximizes downlink opportunities** (more chances the device happens to be in a window/frequency the gateway can serve).
  - Once you catch **one** successful downlink/ACK, immediately push the RX timing fix and then the FSB1 mask.
3. **If neither is possible:**



- Your only remote option is to **wait for a periodic OTAA rejoin** (if the Brown firmware ever does one on its own). Some do, many don't. Until a rejoin happens, the device will keep missing downlinks scheduled at 5s.

#### Quick checks while you try again

- In each `ns.down.data.schedule.*` event, confirm whether `rx1_delay` changed to **1**. If it still says **5**, your fix hasn't been applied.
- After any downlink attempt, look at the **very next uplink** for `f_ctrl.ack: true`. No ack = still deaf.

## Problem Statement

Here's a crisp recap of what we've learned and the playbook to confirm and fix it.

#### What we learned

- **Before outage (8/2 → 8/11):** All 7 devices were ~94% PC Rate.
- **After gateway came back:** 2 devices (TBHH100\_8, TBWL100\_7) returned to ~94%; **5 devices fell to ~34%** and stayed there.
- **Cadence vs loss:** From the TTN log, frame counters kept increasing at the usual pace (e.g., TBWL100\_10 hourly). The devices **kept transmitting**; the **network just didn't hear most of them** → this is network-side loss, not device silence.
- **Downlink path testing:**
  - Raw payload entry in Console sometimes looked "mutated" due to formatter/queue behavior; JSON downlink worked and was clearly traceable.
  - We observed: `as.down.data.receive` (payload accepted) → `as.down.data.forward` (metadata to NS) → `ns.down.data.schedule.success` (scheduled) → `ns.down.transmission.success` (transmitted).
  - The **first uplink after that had no ACK**, and TTN reported `as.down.data.nack.forward ... confirmed_retry.attempt: 3` → device **didn't hear** the downlink.
  - Later uplink from TBWL100\_10 arrived on **903.1 MHz, SF7/BW125 (channel index 4)** with strong link metrics, confirming the device is healthy and still sending.

## Root cause (most likely)

**Primary:** A **MAC-parameter drift** occurred across the outage, leaving **five devices with channel masks and/or receive-window timing that no longer match the gateway**:

- **Channel mask / sub-band mismatch:** Devices likely accepted a **LinkADRReq** (or resumed a prior mask) that moved them to a **different 8-channel US915 sub-band** than the one your RAK7268C listens on; the gateway only hears a subset of their transmissions → **~34% PC Rate**.
- **RX timing mismatch:** We saw confirmed downlinks transmitted but never ACKed. The logs showed **RX1 scheduled at 5s**, while the device likely listens at **1s** (common for 1.0.x). If the device isn't listening when the downlink arrives, it misses MAC corrections (like the channel mask), keeping it out of alignment.

**Why only 2 devices recovered:** Those two probably **kept** the gateway's sub-band and compatible RX timing, so their PC Rate rebounded.

#### How to confirm the diagnosis (quick checks)

1. **Check recent MAC events for a problem device**
  - Look for **LinkADRReq / LinkADRAns, RXTimingSetupReq/Ans, RXParamSetupReq/Ans** around the outage/recovery.
  - Any ...**Ans: success** right before the PC Rate drop is a smoking gun (mask/timing changed on the device).
2. **Spot-check uplink channels**
  - If most uplinks from a problem device land outside your gateway's configured 8-channel set, that's the sub-band mismatch in action.



### 3. Confirmed downlink behavior

- If you see ...schedule.success and ...transmission.success but **no ACK** on the next uplink, timing is off (device missed RX1/RX2).

## Rx Timing Mismatch Detailed Explanation

Query: Earlier you explained the RX timing mismatch as follows: "We saw confirmed downlinks transmitted but never ACKed. The logs showed RX1 scheduled at 5s, while the device likely listens at 1s (common for 1.0.x). If the device isn't listening when the downlink arrives, it misses MAC corrections (like the channel mask), keeping it out of alignment." This doesn't make sense to me. If the device is listening at 1s and downlinks are being sent every 5 seconds, why doesn't the device catch or hear the downlink message?

**Class A devices don't listen continuously.** They only open two **very short receive windows** after *each* uplink. If the network transmits outside those tiny windows, the device can't hear it.

Here's the timing picture:

- **t = 0s:** device sends an uplink.
- **Device's RX windows (what it's actually listening to):**
  - **RX1 at t = 1s** (open only for a few tens/hundreds of milliseconds)
  - **RX2 at t = 2s** (also tiny)
- **What your NS is doing (per your logs):**
  - Scheduling downlinks with **rx1\_delay = 5**, so it transmits at **t = 5s** (RX1) and, if needed, **t = 6s** (RX2).

Because your device is configured to listen at **1s/2s**, by **5s/6s it's already asleep**. So it will **always miss** those downlinks — including all the confirmed retries — which is why you never see an ACK and eventually hit "**max\_retries\_reached**".

Two common gotchas buried in this:

- **"Every 5 seconds" isn't a cadence.** The "5" is an **offset relative to the last uplink**, not a periodic broadcast. Downlinks are only attempted right after an uplink, at the configured offsets.
- **Windows are short.** Even being off by a second guarantees a miss; you have to hit the exact RX1/RX2 slots the device expects.

That's why the RX timing fix is critical: set **Desired Rx1 delay = 1s** (and keep **RX2 = 923.3 MHz / DR8, RX1 offset = 0**), then resend a **confirmed** test downlink and look for **ack: true** on the very next uplink. If you can't deliver that MAC change because the device keeps missing downlinks, you'll need a **rejoin (battery pull)** or a **temporary gateway assist** to catch one downlink and apply the fix.

"NS" = **Network Server** (in your case, The Things Stack's Network Server).

Quick roles map in your logs:

- **GS (Gateway Server):** handles RF from the gateway.
- **NS (Network Server):** LoRaWAN brain — deduplicates uplinks, checks MIC/FCnt, runs ADR, queues MAC commands, **schedules downlinks into RX1/RX2**, and tracks acks.
- **AS (Application Server):** delivers decoded app payloads to your integration (Datacake).

Examples you've shared:

- ns.up.data.receive → NS received an uplink.
- ns.down.data.schedule.success / ns.down.transmission.success → NS scheduled/sent a downlink via the gateway.
- ns.mac.command.unanswered → NS didn't get the expected MAC response from the device.

## Fix strategy (remote-first)

**Order matters: fix RX timing → then fix channel mask.**

1. **Re-establish downlink reliability (RX timing)**



- In TTI **MAC settings** for the device:
    - **RX1 Delay = 1 s** (queues **RXTimingSetupReq**).
    - **RX2 = 923.3 MHz, DR8 (SF12/BW500)** (queues **RXParamSetupReq**).
    - **RX1 DataRate Offset = 0** (stable default).
  - Replace **downlink queue** and send a tiny **confirmed JSON** downlink (e.g., 0x01: AQ==) on **FPort 1**.
  - Verify the 5-event sequence and see the **ACK** on the next uplink.
2. **Apply the correct 8-channel mask (LinkADRReq)**
- Send **LinkADRReq** to enable **only** the channels your RAK7268C uses (your chosen US915 sub-band).
  - Confirm **LinkADRAns (success)** in MAC events.
  - PC Rate should climb back toward **~90–95%** within a day.
3. **Repeat for the other four affected devices.**
- While fixing, consider **temporarily disabling ADR** so TTS doesn't inject new LinkADRReqs that could undo your mask until you're stable.

#### If remote control is limited

- If a device still won't ACK after the RX timing fix, it's probably stuck on the wrong sub-band and missing downlinks entirely.
- In that case, recovery requires **either**:
  - Temporarily broadening/rotating the gateway's sub-band to "recapture" the device and push the fix, **or**
  - A **local power cycle** (OTAA rejoin scans all channels, restoring contact).

#### Hardening / prevention

- **Document and lock** the intended **gateway sub-band** and **device MAC params** (RX1 delay, RX2 DR/freq, RX1 offset).
- **Stage MAC updates explicitly** (RX timing first, then channel masks) and validate ACKs before moving on.
- Consider **ADR off during recovery**, then re-enable once stable.
- Keep a **1-page runbook** (we built this) with JSON downlink template, Base64 table, 5-event checklist, and RX timing steps.

## Deeper Dive into the Sub-band Mismatch Concept

Question: I don't understand the sub-band mismatch hypothesis. Specifically, if the device had moved to a different 8-channel US915 sub-band from what the RAK7268C gateway is listening on, how would that explain only 1/3 of the uplinks making it to the network server. I would have thought that if the device was on the wrong sub-band, that none of the uplinks would make it to the network server.

Here's the overlay explanation with the diagram in mind:

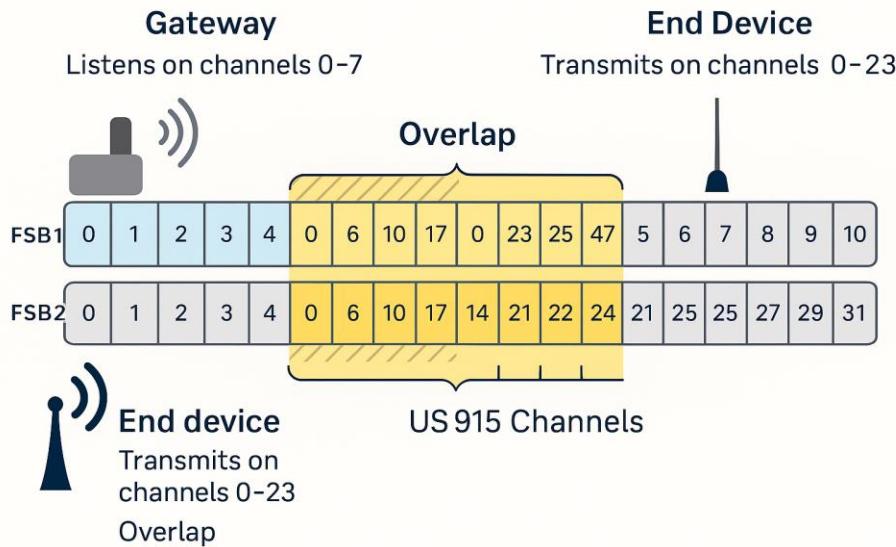


Fig 1 US915 channel blocks with an overlay showing:

- Gateway listens on channels 0–7,
- Device transmits on channels 0–23,
- Overlap = 8/24 = ~33% reception?

### Why ~34% PC Rate

- The gateway (RAK7268C) is fixed to **one 8-channel block** (e.g., channels 0–7).
- The **device** should normally also be limited to those 8 channels by the LinkADRReq channel mask.
- After the outage, some devices appear to have **expanded to transmit across 24 channels** (3 sub-bands).

So:

- Device is sending on 24 total channels.
- Gateway hears only 8 of them.
- **Overlap = 8 ÷ 24 = 33% → matches your observed ~34% PC Rate.**

### Why not 0%?

Because the device's new channel mask still includes the gateway's block, but also includes others. So some transmissions hit the gateway, others miss.

### Why only some devices?

- Two devices stayed with the proper mask (only the gateway's block) → they recovered fully.
- Five accepted or retained a "wide" mask → partial overlap → ~34% reception.

**Refined Root Cause:** After the outage, five devices wound up with **expanded channel masks** (e.g., 24 channels), while the gateway only listens to 8 → producing a consistent ~1/3 reception rate.

**Solution:** Restore RX timing first (so devices hear downlinks), then issue a **LinkADRReq** to force the 8-channel mask matching the gateway.

Would you like me to now draft a **concise executive summary** (1 page) with this final root cause + solution approach, so you can share it with stakeholders without all the technical digressions?



# RAK7268C Configuration

This sections shows a line of inquiry related to how the gateway was set up to FSB1 to see if that shed light on the root cause analysis or the fix plan, but ultimately did not lead to a greater understanding or change in the fix plan – in other words, this line of inquiry only led to more questions, so I paused it.

The screenshot below shows the gateway configuration from the TTI console. Notice that the Frequency plan is FSB 1.

The screenshot displays the TTI console interface for the gateway 'rak7268c-1'. On the left, the 'General information' section shows the Gateway ID as 'rak7268c-1', Gateway EUI as 'AC [REDACTED]', and the Frequency plan as 'United States 902-928 MHz, FSB 1', which is highlighted with a red box. The 'Created at' field shows 'Apr 23, 2022 11:19:30'. On the right, the 'Gateway status' section shows a green bar indicating '30 day uptime' with a value of '63.51ms'. Below this, 'Connection stats' and 'Duty cycle utilization' are listed, both showing 0.00%.

Query: Going back to the fingerprint of a partial channel mask overlap analysis, does it help to know that the gateway has been configured for FSB1?

Yes—it helps a lot, and it also surfaces a red flag.

## What FSB1 means (US915)

- **FSB1 uplink channels:** indices **8–15** ( $\approx 903.9\text{--}905.3\text{ MHz}$  in 200 kHz steps).
- If your gateway truly listens **only** to FSB1 (8 channels), then:
  - Device on **FSB1 only (8 ch)**  $\rightarrow \sim 100\%$  heard.
  - Device on **FSB0+FSB1 (16 ch)**  $\rightarrow 8/16 = 50\%$  heard.
  - Device on **FSB0+FSB1+FSB2 (24 ch)**  $\rightarrow 8/24 \approx 33\%$  heard (your  $\sim 34\%$  fingerprint).
  - Device on a **different single FSB** (e.g., **FSB0**)  $\rightarrow 0\%$  heard.

## But your log shows an uplink at 903.1 MHz (channel index 4)

Channel **4** is in **FSB0 (0–7)**, not FSB1. Since that uplink was received **by your gateway**, this implies one of the following:

1. The gateway is **not actually restricted** to FSB1 and was listening on FSB0 at that moment (config drift or broader channel set).
2. The gateway config/UI says FSB1, but **effective receive mask** is wider (e.g., 16 channels or all 64).
3. There's a mislabel somewhere—but the raw rx\_metadata.frequency=903.1 MHz is pretty definitive.

## How this ties back to the ~34% PC rate

- If the device is hopping over **24 channels** while the gateway effectively hears **only 8** of those, you get **~1/3 reception**—regardless of which specific 8 the gateway is on.
- Your “**FSB1**” claim helps quantify the overlap math, but the **903.1 MHz** reception shows the gateway is (at least sometimes) hearing **FSB0**, too. So either:
  - The gateway’s **actual** active set  $\neq$  the intended FSB1-only set, or
  - It changed over time (or after reboot), or
  - The device (or NS) briefly aligned to a channel the gateway could hear before going wide again.



### What I'd do next (from the logs you already have)

#### 1. Infer the gateway's real uplink mask from data:

Over a recent window, list all **uplink frequencies** observed (rx\_metadata) and map them to channel indices.

- If you see mostly **8 distinct channels** clustered in one block → gateway is truly on one FSB.
- If you see **16 or more** across two/three blocks → gateway is listening wider than intended.

#### 2. For an affected device, count how many distinct channels it uses:

- If it's using ~24 distinct channels while the gateway hears 8 → the **33% fingerprint** is confirmed.

#### 3. Proceed with the fix you're already set up for:

- **RX timing first** (RX1=1s, RX2=923.3/DR8, Offset=0 → verify ACK on a tiny confirmed downlink).
- Then push **LinkADRReq** to constrain the device to the **gateway's actual 8-channel block**.

If you want, I can draft a tiny script plan (or Excel formula approach) to turn your TTN log into two quick tables:

- **Observed gateway channel set** (unique frequencies + counts), and
- **Per-device channel spread** (unique channels used + overlap estimate).

## Implementing the Rx Timing Fix

### Plan

- In **TTI MAC settings** for the device:
  - **RX1 Delay = 1 s** (queues **RXTimingSetupReq**).
  - **RX2 = 923.3 MHz, DR8 (SF12/BW500)** (queues **RXParamSetupReq**).
  - **RX1 DataRate Offset = 0** (stable default).
- Replace **downlink queue** and send a tiny **confirmed** JSON downlink (e.g., 0x01: AQ==) on **FPort 1**.
- Verify the 5-event sequence and see the **ACK** on the next uplink.

Here's a clean JSON example you can paste into the **TTI Console** downlink scheduling box. This will send a tiny **confirmed downlink** with payload 0x01 (Base64: AQ==) on **FPort 1**:

```
{  
  "downlinks": [  
    {  
      "frm_payload": "AQ==",  
      "f_port": 1,  
      "confirmed": true,  
      "priority": "HIGH"  
    }  
  ]  
}
```

### Explanation of Fields

- **frm\_payload**: "AQ==" → Base64 encoding of 0x01.
- **f\_port**: 1 → Application port 1 (safe test port for the TBWL100).
- **confirmed**: true → Device must ACK this downlink in the next uplink.
- **priority**: "HIGH" → Ensures it gets scheduled promptly.

### Solution Approach

#### 1. Fix RX Timing (first priority)

- RX1 Delay = 1s



- RX2 = 923.3 MHz / DR8 (SF12, 500 kHz)
  - RX1 Offset = 0
  - Send a small confirmed downlink (0x01) and verify ACK.
2. **Re-align Channel Mask**
- Once ACKs are working, send **LinkADRReq** to restrict the device to the gateway's 8-channel block.
  - Confirm success via **LinkADRAAns** in MAC logs.
3. **Monitor PC Rate**
- Expect recovery from ~34% → ~90–95% within 24h.
  - Repeat for each affected device.

### What to Look for in Console

After you send this:

1. as.down.data.receive → Console accepted the JSON.
2. as.down.data.forward → Forwarded to NS.
3. ns.down.data.schedule.success → Scheduled for gateway transmission.
4. ns.down.transmission.success → Confirmed transmitted.
5. **Next uplink from device should include ACK** → proves downlink path is working.

Query: I want to proceed with the RX timing fix but am having trouble matching up the instructions you gave me ((RX1=1s, RX2=923.3/DR8, Offset=0) with the following options that I see in the TTI console under Advanced MAC Settings for the device: Desired Rx1 delay (currently set to 5 sec), Desired Rx1 data rate offset (currently set to 0), Desired Rx2 data rate index (currently set to 8), Desired Rx2 frequency (currently set to 923.3 MHz), Desired maximum duty cycle (currently set to 100%), Status count periodicity (currently set to 200 messages), Status time periodicity (currently set to 86400 seconds), Adaptive data rate (currently set to Dynamic mode), and ADR margin (currently set to 15 dB), and Use default settings for number of retransmissions (which is set for 3 max tries).

Response:

Perfect — you're looking at exactly the right section in the TTI Console. Let me map my earlier shorthand (RX1=1s, RX2=923.3/DR8, Offset=0) to the **fields you see in Advanced MAC Settings**:

---

#### 1. Desired Rx1 delay

- **Current:** 5 sec
- **Change to: 1 sec**
- Why: This sets how long after the uplink the device listens for RX1 downlink. Your device is almost certainly expecting 1 second, not 5. This is the key fix.

---

#### 2. Desired Rx1 data rate offset

- **Current:** 0
- **Leave as: 0**
- Why: This ensures the RX1 downlink data rate directly maps from the uplink DR. Zero offset is the stable choice for US915.

---

#### 3. Desired Rx2 data rate index

- **Current:** 8
- **Leave as: 8**
- Why: DR8 = SF12 / 500 kHz (the LoRaWAN default RX2 window for US915).

---

#### 4. Desired Rx2 frequency

- **Current:** 923.3 MHz



- **Leave as: 923.3 MHz**
- Why: This is the LoRaWAN default RX2 frequency for US915.

## 5. Other fields

- **Desired maximum duty cycle:** Leave at 100%.
- **Status count periodicity:** Leave as is.
- **Status time periodicity:** Leave as is.
- **Adaptive data rate (ADR):** For now, you may leave on “Dynamic mode.” If you run into continued MAC drift, you can switch it off temporarily, but it isn’t the immediate cause of your ~34% PC Rate.
- **ADR margin:** Leave at 15 dB.
- **Number of retransmissions:** Leave at defaults.

### Action Summary

In your Console:

1. Go to **Advanced MAC Settings** for TBWL100\_10.
2. Change only:
  - **Desired Rx1 delay → 1 sec.**
3. Leave all other fields as they are.
4. Save / queue this change.
5. Then send a **tiny confirmed test downlink** (the JSON AQ== we prepared).
6. Watch the uplink: look for **ack: true** in the next uplink.

**Key point:** Fix RX1 timing first. Once you see ACKs working, you can safely proceed to push the **LinkADRReq channel mask fix** for the gateway’s FSB1 block.

Would you like me to also write out the **exact LinkADRReq settings for FSB1 (channels 8–15)** so you can queue that step next once the ACK test succeeds?

## Executing the Fix

### Before Change

Here is what the network setting look like for TBWL1--\_10 before I attempt the recommended changes.



TBWL100-10

ID: eui-e8e1e1000104defb

 Add label

 Last activity 3 hours ago • ↑ 26,119 up / 1,610 (Nwk) down



## Network layer

## LoRaWAN network-layer settings, behavior and session

[Collapse](#)

## Frequency plan

United States 902-928 MHz, FSB 1

LoRaWAN version ⑦ \*

LoRaWAN Specification 1.0.3

Regional Parameters version ⑦ \*

BP001 Regional Parameters 1.0.3 revision A

## LoRaWAN class capabilities

- Supports class B
  - Supports class C

## Activation mode

- Over the air activation (OTAA)
  - Activation by personalization (ABP)
  - Define multicast group (ABP & Multicast)

Device address ? \*

26 09 2B CA

 Generate

NwkSKey ? \*

 Generate

## Session and MAC state reset

### Reset session and MAC state



Session and MAC state reset ⓘ

[Reset session and MAC state](#)[Advanced MAC settings ^](#)

Frame counter width ⓘ

 16 bit  32 bit

Desired Rx1 delay ⓘ

5  sec

Desired Rx1 data rate offset ⓘ

0 

Desired Rx2 data rate index ⓘ

8 

Desired Rx2 frequency ⓘ

923.3  MHz 

Desired maximum duty cycle ⓘ

100% 

Factory preset frequencies ⓘ

[+ Add Frequency](#)

List of factory-preset frequencies. Note: order is respected.

Status count periodicity ⓘ

200  messages

Status time periodicity ⓘ

86400  seco... 

Adaptive data rate (ADR) ⓘ

- Dynamic mode
- Static mode
- Disabled

ADR margin ⓘ

15  dB Use default settings for number of retransmissions ⓘ[Save changes](#)



## Change

TBWL100-10

ID: eui-e8e1e1000104defb

+ Add label

⌚ Last activity 3 hours ago • ⬆ 26,119 up / 1,610 (Nwk) down



...

Generate

Session and MAC state reset ⓘ

Reset session and MAC state

Advanced MAC settings ^

Frame counter width ⓘ

 16 bit  32 bit

Desired Rx1 delay ⓘ

1  sec

Desired Rx1 data rate offset ⓘ

0 

Desired Rx2 data rate index ⓘ

8 

Desired Rx2 frequency ⓘ

923.3  MHz 

Desired maximum duty cycle ⓘ

100% 

Factory preset frequencies ⓘ

+ Add Frequency

List of factory-preset frequencies. Note: order is respected.

Status count periodicity ⓘ

200  messages

Status time periodicity ⓘ

86400  seco... 

Adaptive data rate (ADR) ⓘ

- Dynamic mode
- Static mode
- Disabled

ADR margin ⓘ

15  dB Use default settings for number of retransmissions ⓘ

Save changes



**TBWL100-10**  
ID: eui-e8e1e1000104defb

+ Add label      Last activity 4 hours ago • ↑ 26,119 up / 1,610 (Nwk) down

Session and MAC state reset  

**Reset session and MAC state**

Advanced MAC settings ^

Frame counter width 

16 bit  32 bit

Desired Rx1 delay 

1   sec

Desired Rx1 data rate offset 

0  

Desired Rx2 data rate index 

8  

Desired Rx2 frequency 

923.3  MHz 

Desired maximum duty cycle 

100% 

Factory preset frequencies 

+ Add Frequency

List of factory-preset frequencies. Note: order is respected.

Status count periodicity 

200  messages 

Status time periodicity 

86400  sec... 

Adaptive data rate (ADR) 

Dynamic mode  
 Static mode  
 Disabled

ADR margin 

15  dB 

Use default settings for number of retransmissions 

**Save changes**



eui-e8e1e1000104defb  
End device updated

Application layer



Sept. 25, 2025

Applications > Application for Hitechdb's Long... > End devices > TBWL100-10 > Live data

TBWL100-10  
ID: eui-e8e1e1000104defb + Add label

Last activity 3 hours ago • 26,119 up / 1,610 (Nwk) down

Device overview Live data Messaging Location Payload formatters Settings

TIME TYPE DATA PREVIEW

Verbose stream Export as JSON Clear

08:20:41 Update end device [ "supports\_class\_c", "supports\_class\_b", "mac\_settings.status\_count\_periodicity", "mac\_settings.status\_time\_periodicity", "mac\_settings.adr", "mac\_settings.adr\_margin", "mac\_settings.use\_adr", "mac\_settings.desired\_max\_duty\_cycle", "mac\_settings.supports\_32\_bit\_f\_cnt", "mac\_settings.desired\_rx2\_frequency", "mac\_settings.desired\_rx2\_data\_rate\_index", "mac\_settings.desired\_rx1\_data\_rate\_offset", "mac\_settings.desired\_rx1\_delay" ]

08:03:05 Console: Stream reconnected The stream connection has been re-established

08:02:58 Console: Stream connection closed The connection was closed

07:58:08 Console: Filter disabled The previously set event filter has been disabled

04:35:38 Fail to send webhook

04:35:33 Schedule data downlink for transmission

04:35:33 Forward uplink data message

04:35:33 Successfully processed data message

03:35:44 Fail to send webhook

03:35:39 Schedule data downlink for transmission

03:35:39 Forward uplink data message

03:35:39 Successfully processed data message

```
{
  "name": "ns.end_device.update",
  "time": "2025-09-25T12:20:41.266020747Z",
  "identifiers": [
    {
      "device_ids": {
        "device_id": "eui-e8e1e1000104defb",
        "application_ids": {
          "application_id": "hitechdb-longview-datacake"
        }
      }
    }
  ],
  "data": {
    "@type": "type.googleapis.com/google.protobuf.Value",
    "value": [
      "supports_class_c",
      "supports_class_b",
      "mac_settings.status_count_periodicity",
      "mac_settings.status_time_periodicity",
      "mac_settings.adr",
      "mac_settings.adr_margin",
      "mac_settings.use_adr",
      "mac_settings.desired_max_duty_cycle",
      "mac_settings.supports_32_bit_f_cnt",
      "mac_settings.desired_rx2_frequency",
      "mac_settings.desired_rx2_data_rate_index",
      "mac_settings.desired_rx1_data_rate_offset",
      "mac_settings.desired_rx1_delay"
    ]
  },
  "correlation_ids": [
    "rpc:/ttn.lorawan.v3.NsEndDeviceRegistry/Set:b55b5551-6ef5-475e-bc9e-71ee5dac8516"
  ]
}
```



```
],
"origin": "ip-10-22-15-21.us-west-1.compute.internal",
"context": {
  "tenant-id": "CghkYXRhY2FrZQ=="
},
"visibility": {
  "rights": [
    "RIGHT_APPLICATION_DEVICES_READ"
  ]
},
"authentication": {
  "type": "Bearer",
  "token_type": "AccessToken",
  "token_id": "P6XFM4NQBHY225EXZY6HPNDDL4MXY7R47C2WZCA"
},
"remote_ip": "73.207.72.127",
"user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:143.0) Gecko/20100101 Firefox/143.0",
"unique_id": "01K60D33NJP5D1ZZS7TY0VGVD8"
}
```



Applications > Application for Hitechdb's Long... > End devices > TBWL100-10 > Messaging + Add Star Share Log

**TBWL100-10**  
ID: eui-e8e1e1000104defb Last activity 4 hours ago • ↓ 26,119 up / 1,610 (Nwk) down Star More

Device overview Live data **Messaging** Location Payload formatters Settings

Schedule downlink Simulate uplink

**Schedule downlink**

Insert Mode Replace downlink queue   Push to downlink queue (append)

FPort\* 1

Payload type  Bytes  JSON

**Payload**

```
1 {
2   "downlinks": [
3     {
4       "frm_payload": "AQ==",
5       "f_port": 1,
6       "confirmed": true,
7       "priority": "HIGH"
8     }
9   ]
10 }
11 |
```

The decoded payload of the downlink message

Confirmed downlink

**Schedule downlink**



Sept. 25, 2025

Applications > Application for Hitchedb's Long... > End devices > TBWL100-10 > Live data

TBWL100-10  
ID: eui-e8e1e1000104defb + Add label

Last activity 4 hours ago • ↑ 26.119 up / 1.610 (Nwk) down

Device overview Live data Messaging Location Payload formatters Settings

TIME TYPE DATA PREVIEW

↓ 08:25:11 Forward downlink data message FPort: 1

↓ 08:25:11 Receive downlink data message Payload: { downlinks: [...] } FPort: 1

⌚ 08:20:41 Update end device [ "supports\_class\_c", "supports\_class\_b", "mac\_settings.status\_count\_periodicity", "mac\_settings.status\_time\_periodicity", "mac\_settings.adr", "mac\_settings.adr\_margin", "mac\_settings.use\_adr", "mac\_sett

ⓘ 08:03:05 Console: Stream reconnected The stream connection has been re-established

ⓘ 08:02:58 Console: Stream connection closed The connection was closed

ⓘ 07:58:08 Console: Filter disabled The previously set event filter has been disabled

ⓘ 04:35:38 Fail to send webhook

↓ 04:35:33 Schedule data downlink for transmissi...

↑ 04:35:33 Forward uplink data message

↑ 04:35:33 Successfully processed data message

ⓘ 03:35:44 Fail to send webhook

↓ 03:35:39 Schedule data downlink for transmissi...

↑ 03:35:39 Forward uplink data message

↑ 03:35:39 Successfully processed data message

```
{  
  "name": "as.down.data.receive",  
  "time": "2025-09-25T12:25:11.238365983Z",  
  "identifiers": [  
    {  
      "device_ids": {  
        "device_id": "eui-e8e1e1000104defb",  
        "application_ids": {  
          "application_id": "hitechdb-longview-datacake"  
        },  
        "dev_eui": "E8E1E1000104DEFB",  
        "join_eui": "E8E1E10001013641"  
      }  
    }  
  ],  
  "data": {  
    "@type": "type.googleapis.com/ttn.lorawan.v3.ApplicationDownlink",  
    "f_port": 1,  
    "decoded_payload": {  
      "downlinks": [  
        {  
          "confirmed": true,  
          "f_port": 1,  
          "frm_payload": "AQ==",  
          "priority": "HIGH"  
        }  
      ]  
    },  
    "confirmed": true,  
    "correlation_ids": [  
      "as:downlink:01K60DBBA5PP5X5YR5W6HX4M48",  
      "rpc:/ttn.lorawan.v3.AppAs/DownlinkQueueReplace:22e47788-7476-41d7-a0cb-22d6893cee6c"  
    ]  
  }  
}
```



```
],
  "confirmed_retry": {
    "attempt": 1
  }
},
"correlation_ids": [
  "as:downlink:01K60DBBA5PP5X5YR5W6HX4M48",
  "rpc:/ttn.lorawan.v3.AppAs/DownlinkQueueReplace:22e47788-7476-41d7-a0cb-22d6893cee6c"
],
"origin": "ip-10-22-15-114.us-west-1.compute.internal",
"context": {
  "tenant-id": "CghkYXRhY2FrZQ=="
},
"visibility": {
  "rights": [
    "RIGHT_APPLICATION_TRAFFIC_READ"
  ]
},
"authentication": {
  "type": "Bearer",
  "token_type": "AccessToken",
  "token_id": "P6XFM4NQBHY225EXZY6HPNDDL4MXY7R47C2WZCA"
},
"remote_ip": "73.207.72.127",
"user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:143.0) Gecko/20100101 Firefox/143.0",
"unique_id": "01K60DBBA6NBHJJ7G93GFXDG9P"
}
```

## After Change

After performing the Rx Timing configuration change for the device in the TTI console and queuing up a small downlink message, we wait and monitor the Live Message logs.....

### 1. Fix RX Timing (first priority)

- RX1 Delay = 1s
- RX2 = 923.3 MHz / DR8 (SF12, 500 kHz)
- RX1 Offset = 0
- Send a small confirmed downlink (0x01) and verify ACK.

The screenshot below shows the live data message log as we being our waiting period...



Applications > Application for Hitechdb's Long... > End devices > TBWL100-10 > Live data

TBWL100-10  
ID: eui-e8e1c1000104defb + Add label

Last activity 4 hours ago • ↑ 26.119 up / 1,610 (Nwk) down

Device overview Live data Messaging Location Payload formatters Settings

DATA PREVIEW

TIME TYPE

↓ 08:25:11 Forward downlink data message FPort: 1

↓ 08:25:11 Receive downlink data message Payload: { downlinks: [...] } FPort: 1

⟳ 08:20:41 Update end device [ "supports\_class\_c", "supports\_class\_b", "mac\_settings.status\_count\_periodicity", "mac\_settings.status\_time\_periodicity", "mac\_settings.adr", "mac\_settings.adr\_margin", "mac\_settings.use\_adr", "mac\_sett

⌚ 08:03:05 Console: Stream reconnected The stream connection has been re-established

⌚ 08:02:58 Console: Stream connection closed The connection was closed

⌚ 07:58:08 Console: Filter disabled The previously set event filter has been disabled

⌚ 04:35:38 Fail to send webhook

↓ 04:35:33 Schedule data downlink for transmissi...

↑ 04:35:33 Forward uplink data message

↑ 04:35:33 Successfully processed data message

⌚ 03:35:44 Fail to send webhook

↓ 03:35:39 Schedule data downlink for transmissi...

↑ 03:35:39 Forward uplink data message

↑ 03:35:39 Successfully processed data message

Here's the **5-event sequence** to watch for in the TTI Console Live Data when verifying the **RX timing fix** with your tiny confirmed test downlink:

### Confirmed Downlink Verification Path

1. **as.down.data.receive**
  - Application Server accepted your JSON downlink request.
  - Check that f\_port and frm\_payload match what you sent (e.g., AQ== for 0x01).
2. **as.down.data.forward**
  - Application Server forwarded the downlink metadata to the Network Server.
  - If payload crypto is handled by the NS, the FRMPayload itself may not appear — that's normal.
3. **ns.down.data.schedule.success**
  - Network Server successfully scheduled the downlink for transmission to the gateway.
  - Look for the correct gateway\_id (your RAK7268C).
4. **ns.down.transmission.success**
  - Confirms the gateway actually transmitted the downlink on-air.
  - This means the RF frame was sent during RX1/RX2.
5. **Next uplink contains ACK**
  - In the uplink JSON, check f\_ctrl.ack: true.
  - Or look for an explicit event like "*Confirmed downlink acknowledged by device.*"
  - This proves the device received and acknowledged the confirmed downlink.

Below – we just got step 2



Sept. 25, 2025

The screenshot shows the HitechDB interface with the device ID TBWL100-10. The 'Live data' tab is selected, displaying a log of events. A red arrow points from the log entry '↓ 08:35:11 Forward downlink data message FPort: 1' to a modal window titled 'EVENT DETAILS'. The modal contains a JSON object representing the event. Another red arrow points from the JSON object to a comment in the interface: 'Yes — the JSON you pasted is exactly step 2 in the 5-event sequence'.

```

{
  "name": "as.down.data.forward",
  "time": "2025-09-25T12:28:11.247694542Z",
  "unique_id": "01K60DBBAF12FTH051SHNKERB"
}
  
```

**Step Mapping**

1. as.down.data.receive → NS accepted your downlink request (not shown in your snippet, but should appear just before this)
2. as.down.data.forward → This is the event you just showed. The Application Server has forwarded the downlink (with f\_port: 1, confirmed: true, f\_cnt: 1611) to the Network Server.
3. Next you want to see: ns.down.data.schedule.success (NS successfully scheduled to your RAK gateway).
4. Then: ns.down.transmission.success (gateway actually transmitted it on-air).
5. Finally: The next uplink from the device should contain ack: true (or an explicit "Confirmed downlink acknowledged" event).

So right now you're at event 2 of 5. Keep watching the Live Data log — if you see the ns.down.data.schedule.success and ns.down.transmission.success follow, that confirms the network sent it. The critical final check is the uplink ACK: if that's missing, the device still isn't aligned on RX timing.

Do you want me to show you how to spot the ACK in the uplink JSON so you can quickly verify the device really heard and responded?

Looking at the log screenshot below, it appears as if the device is not accepting the downlink message and the network server keeps resending it

The screenshot shows the HitechDB interface with the device ID TBWL100-10. The 'Live data' tab is selected, displaying a log of events. A red box highlights a sequence of events: 'Forward downlink sent' (DevAddr: 26 09 2B CA, FPort: 1), 'Downlink successfully transmitted' (DevAddr: 26 09 2B CA, FPort: 1, Rx1 Delay: 5), and 'Forward downlink not acknowledged' (DevAddr: 26 09 2B CA, FPort: 1). This sequence repeats multiple times throughout the log.



This is the Network Server trying to schedule a confirmed downlink to your device. It's step 3 (attempt) in the downlink chain.

### What it tells you

- Event: `ns.down.data.schedule.attempt` → NS is attempting to place a downlink into an RX window.
- Frame type: `CONFIRMED_DOWN` (device must ACK).
- FPort: `1` (your tiny test/app payload).
- FOpts: `"Bg=="` (1 byte of MAC command in the header — consistent with NS retrying a MAC like `DevStatusReq`).
- RX1 params:
  - `rx1_delay: 5` ← still 5 seconds (this is your problem)
  - `rx1_frequency: 925700000`
  - `rx1_data_rate: BW500, SF7` (typical US915 RX1 mapping)
- RX2 defaults shown: `923.3 MHz, DR8`.
- Priority: `HIGHEST`.
- Frequency plan: `US_902_928_FSB_1`.

### How to read it in your workflow

- This means the NS is doing the right thing—trying to deliver your confirmed downlink (with a small app payload + a MAC in FOpts).
- But because RX1 Delay is still 5s, and your device likely listens at 1s, it will miss this downlink.
- If it misses, you'll typically see:
  - `ns.down.data.schedule.success` (or not, depending on timing)
  - `ns.down.transmission.success` (gateway actually transmitted)
  - Then no ACK on the next uplink and possibly `as.down.data.nack.forward` and `ns.mac.command.unanswered`.

### Bottom line

This event confirms the NS is trying to send your confirmed downlink, but the RX1=5s setting is still in effect.

Change Desired Rx1 delay → 1 sec, keep RX2 = 923.3 / DR8, then resend your tiny confirmed JSON downlink. The next uplink should show `ack: true` once timing is aligned.

## Implementing the push **LinkADRReq** to constrain the device to the gateway's actual 8-channel block.

Note that we never got to this step since we couldn't complete the Rx Timing step.

### Plan

Query: Would you like me to also write out the **exact LinkADRReq settings for FSB1 (channels 8–15)** so you can queue that step next once the ACK test succeeds?

Response:



Totally. Here's exactly how to set **FSB1 (channels 8–15)**—both the **safe Console way** and the **explicit LinkADRReq fields** if you need them.

---

#### Option A (recommended): Use the Console's Channels UI

TTI Console → End device → MAC settings → Advanced → Channels

- Choose **Custom channel mask** (or the “Edit channels” dialog).
- **Enable only channels 8–15** (these are the FSB1 uplink channels: 903.9–905.3 MHz).
- **Disable 0–7 and 16–63**.
- Click **Queue MAC** (this will auto-generate the required sequence of LinkADRReq commands for you).
- Use **Replace downlink queue**, Confirmed = **Yes**.
- After it applies, look for **LinkADRAbs (success)** in MAC events.

This UI route is safest—TTI builds the right multi-command sequence automatically.

---

#### Option B (explicit LinkADRReq values)

If you're entering **LinkADRReq** manually, US915 masks are applied in 16-channel blocks using **ChMaskCntl**. To force **FSB1 only (8–15)** and disable all others:

1. **Block 0 (channels 0–15): enable only 8–15**
  - **ChMaskCntl = 0**
  - **ChMask = 0xFF00** (bits 8–15 = 1, bits 0–7 = 0)
  - (Leave **DataRate**, **TxPowerIdx**, **NbTrans** unchanged unless you intend to modify them.)
2. **Block 1 (channels 16–31): disable all**
  - **ChMaskCntl = 1**
  - **ChMask = 0x0000**
3. **Block 2 (channels 32–47): disable all**
  - **ChMaskCntl = 2**
  - **ChMask = 0x0000**
4. **Block 3 (channels 48–63): disable all**
  - **ChMaskCntl = 3**
  - **ChMask = 0x0000**

Queue these **in order**, using **Replace downlink queue**. Watch for **LinkADRAbs** responses—each should come back **success**.

---

#### After applying the mask

1. Send your tiny **confirmed** test downlink (JSON AQ== on FPort 1).
2. Verify the **ACK** on the next uplink.
3. Monitor PC Rate (should climb from ~34% toward ~90–95%).

If you'd like, I can also give you the same **explicit masks** for **FSB0, FSB2...FSB7** so you have them handy.