ChatGPT

Setting Up a Point-to-Point LoRa Link (RN2903 on Manjaro Laptop and Raspberry Pi)

Setting up a direct LoRa communication between your Manjaro Linux laptop and Raspberry Pi is feasible using the Microchip RN2903 LoRa modules. Unlike Wi-Fi or Bluetooth, LoRa offers **long-range, low-bandwidth** links (15+ km in open areas, ~5 km in urban conditions [1]) at the cost of lower data rate. This makes LoRa ideal for distant device-to-device links where Wi-Fi Ad-hoc or Bluetooth would fail due to range. Below, we outline how to establish a **point-to-point LoRa connection** (no gateway, just two nodes) to perform a simple "ping" test and transfer a file. We'll use the RN2903's built-in **ASCII command interface** over UART (similar to AT commands) to configure the radios and send/receive data [2] [3].

Hardware Setup and Prerequisites

1. **Connect the RN2903 USB Dongles:** Plug each RN2903 PICtail module into the USB port of the laptop and the Raspberry Pi. The PICtail board has an on-board USB-to-UART bridge, so it will appear as a serial COM port (e.g. `/dev/ttyACM0` or `/dev/ttyUSB0` on Linux) [4] [5].

2. **Important:** Attach the **915 MHz antenna** to each RN2903's SMA connector *before* powering it via USB [6]. Transmitting without an antenna can damage the module's RF amplifier.

3. **Serial Interface Configuration:** Open a serial terminal program on both the laptop and Pi (e.g. `minicom`, `screen`, or PuTTY). Use the settings **57600 baud, 8 data bits, no parity, 1 stop bit (8N1)**, which is the RN2903's default UART configuration [7]. Once connected, you can type commands and should see responses from the module. For example, try the command:

   ```
   sys get ver
   ```

   This should return the firmware version string if communication is working (e.g. "`RN2903 x.y.z ...`") [8].

4. **Understand Command Modes:** The RN2903 accepts simple ASCII text commands (no `AT+` prefix) ended by newline. Commands are grouped into three categories [9]:

5. `sys` – system commands (device info, resets, etc.).
6. `mac` – LoRaWAN MAC-layer commands (joining network, sending via LoRaWAN protocol).
7. `radio` – direct radio transceiver commands (for raw LoRa or FSK transmission, useful for point-to-point).
   For direct device-to-device LoRa, we will **bypass LoRaWAN** and use `radio` commands.

Configuring the RN2903 Modules for P2P LoRa

By default, the RN2903 is geared for LoRaWAN networks, so we need to put it into a mode where we can send raw LoRa packets. The **LoRaWAN stack must be disabled (paused)** before using raw radio commands [10] [11]:

- **Pause LoRaWAN on each module:** In the terminal for each device, enter:

```
mac pause
```

  This command suspends the LoRaWAN protocol functionality, freeing the radio for direct use. It will respond with a number (the number of milliseconds the MAC will stay paused – typically a very large value) [10]. *Note:* You **must** do this before any `radio tx` or `radio rx` command; even if you never joined a LoRaWAN network, the RN2903 requires `mac pause` to use the radio directly [11].

- **Set both modules to the same LoRa parameters:** To communicate point-to-point, **both radios must share the same frequency and modulation settings**. You can query defaults with `radio get ...` commands, but it's safest to explicitly set them on both sides:

- **Frequency:** Choose a frequency in the 915 MHz ISM band. For example, set **915.0 MHz** on each:

```
radio set freq 915000000
```

  (The RN2903 supports 902–928 MHz [12]. Avoid frequencies actively used by local LoRaWAN networks, to prevent interference [13].)
- **Spreading Factor (SF):** This controls range vs data rate. A higher SF (like SF12) gives longer range and more sensitivity but slower transmission; lower SF (SF7) is faster but shorter range. For maximum range (and to match factory default), use **SF12** on both:

```
radio set sf sf12
```

  (Valid values: `sf7`–`sf12` [14]. RN2903 default is typically SF12 [15].)
- **Bandwidth:** Use **125 kHz** (LoRaWAN standard bandwidth):

```
radio set bw 125
```

  (Options are 125, 250, or 500 kHz [16]. 125 kHz gives best sensitivity [17].)
- **Coding Rate:** Set the forward error correction coding to **4/5** (LoRaWAN default):

```
radio set cr 4/5
```

  (Other options: 4/6, 4/7, 4/8 [18]. 4/5 offers maximum error correction [19].)

- **Sync Word (Optional):** LoRa uses a sync word to distinguish networks. The default sync word is 0x34 (used by LoRaWAN public networks). For a private point-to-point link, you can set a different sync word (e.g. 0x12) on both modules so they ignore LoRaWAN traffic:

  ```
  radio set sync 12
  ```

  This isn't strictly required if you chose a unique frequency, but it helps ensure only your two nodes talk to each other.
- **CRC:** Enable cyclic redundancy check (if not already on) so packets are integrity-checked:

  ```
  radio set crc on
  ```

  (By default the RN2903 should have CRC on; LoRaWAN uses CRC.)

- **Transmit Power:** Set a transmit power level. The RN2903 supports 2 to 20 dBm output [20]. For initial tests at close range, you can use a moderate power like 5 dBm to avoid RF saturation; for longer range tests, use **14 dBm** or higher. For example:

  ```
  radio set pwr 14
  ```

  This sets ~14 dBm (25 mW). The module can go up to 20 dBm (100 mW) if allowed in your region [20]. *(Note: In some regions like EU, 14 dBm is the legal limit on certain frequencies. In the US 915 MHz band, up to 30 dBm is allowed with hopping spread spectrum, but 20 dBm is the module max. Always ensure you comply with local regulations on transmit power and duty cycle [21] [22].)*

- **Verify settings (optional):** You can use `radio get freq`, `radio get sf`, etc., to confirm each parameter [23] [24]. At this point, both radios are configured identically and ready to communicate on the chosen channel.

Performing a "Ping" Test over LoRa

With the modules configured, you can send a test message from one to the other to simulate a "ping." Because LoRa is not IP-based, this **ping** will be a simple user-defined message and response:

1. **Set one node to receive:** On the Raspberry Pi's RN2903 (for example), enter receive mode by typing:

   ```
   radio rx 0
   ```

   The parameter `0` puts the radio in **continuous receive mode** (no timeout) [25] [26]. The module will reply `ok` to indicate it's now listening [27]. It will stay in RX mode until it receives a LoRa packet. *(Tip: "0" means wait indefinitely; you could use a number of symbols or milliseconds for a finite RX window, but continuous mode is simplest for a ping test.)*

2. **Transmit from the other node:** On the laptop's RN2903, send a LoRa packet with a small payload. The `radio tx` command takes a **hexadecimal string** as the data payload [28]. For example, to send the ASCII text "PING", you would provide its hex representation (`50494E47`):

   ```
   mac pause         (if not already paused on this device)
   radio tx 50494E47
   ```

   After entering `radio tx ...`, you should see `ok` (command accepted) followed by `radio_tx_ok` once the packet is transmitted [29]. On the sender's side, the sequence looks like:

```
mac pause
4294967245            <-- (pause response)
radio set pwr 14
ok
radio tx 50494E47     <-- (send "PING")
ok
radio_tx_ok           <-- (transmission completed)
```

(The `mac pause` and power-set commands are shown for completeness – you already did those in setup. The key line is `radio tx ...`.)

1. **Receive the packet:** Over on the Pi's terminal (the side that is in `radio rx 0` mode), as soon as the packet is received, the RN2903 will print a line with the received data:

   ```
   radio_rx 50494E47
   ```

   This indicates a successful reception and shows the payload in hex (in this case `50494E47`). You can translate that from hex to ASCII to verify it spells "PING". The RX side's console would have shown something like:

```
radio rx 0
ok
radio_rx 50494E47     <-- (packet received)
```

If no packet was received (within a timeout, if one was set), the module would respond with `radio_err` instead [30] [31]. In continuous mode, it will just wait indefinitely until a packet arrives.

1. **Reply back (Pong):** To complete a ping-pong demonstration, you can now have the receiver send a response. For instance, on the Pi (after seeing the `radio_rx`), you could issue:

   ```
   radio tx 504F4E47
   ```

(which is "PONG" in hex). Then quickly put the laptop's module into RX mode (`radio rx 0`) to listen for it. If timed right, the laptop side will output `radio_rx 504F4E47` upon receiving the pong. In practice, coordination is needed because each module cannot receive while it's transmitting. A simple way to handle this is to **script the exchange** (have the sender wait briefly then listen for a response). But manually, you can still observe each direction one at a time to confirm both modules can send and receive.

The above steps demonstrate a basic **LoRa ping**: one node sends a message and the other receives (and optionally responds). The Microchip RN modules are essentially acting like transparent LoRa modems using these commands. The example from a real test is shown below, where one device was set to receive and another sent a payload (here a hex string `123456789012`):

One device (receiver) commands:
`mac pause`
`radio rx 0`
...Receiver prints: `ok` (listening)

Other device (transmitter) commands:
`mac pause`
`radio set pwr 14`
`radio tx 123456789012`
...Transmitter prints: `ok` then `radio_tx_ok` (sent)

Receiver then prints: `radio_rx 123456789012` (payload received) [32] [33]

As you can see, the process is straightforward but **half-duplex** – only one side sends at a time while the other listens. There is no built-in "ping" utility like an ICMP ping; it's up to us to send a message and possibly code the other side to reply.

Tip: For repeated tests or automation, consider writing a small Python script (using PySerial) on each device to handle send/receive logic. For example, you could have the Pi's script continuously listen and whenever it receives data, it prints it and responds with a predefined acknowledgment. The laptop's script could send a ping, wait for a `radio_rx` response, and measure the round-trip time. This would give you a **round-trip latency** measurement similar to a ping (note that LoRa latency will be much higher and variable compared to Wi-Fi/Bluetooth due to low bitrate and duty cycle delays).

Transferring a File via LoRa

Transferring a file between the two devices over LoRa is possible, but keep in mind the **bandwidth is very limited** (e.g. ~300 bps to a few kbps depending on SF and BW). This will be **much slower than Wi-Fi or Bluetooth**, but it will work over long distances. Here's how you can send a file:

1. **Prepare the file data:** You'll need to send the file in chunks that fit into LoRa packets. The RN2903 supports up to 255 bytes per `radio tx` in LoRa mode [28]. It's wise to use smaller chunks (e.g. 50–100 bytes) to increase chances of success and comply with airtime limits. Convert each chunk of the file into a hex string. For example, if you have a binary file or text, you can use a utility or script to

read it and output hex. (On Linux, `xxd -p` can do this, or a Python script can do `.hex()` on bytes.)

2. **Send chunks sequentially:** Send each chunk with a `radio tx <hexdata>` command. After each send, wait for the module to return `radio_tx_ok` before sending the next chunk [34]. **Throttle your sends** to avoid overlapping or violating duty cycle – it's good to pause a bit between packets. For SF12 at 125 kHz, a full 255-byte packet can take on the order of 1-2 seconds on air, and regulations (in some regions) limit duty cycle or impose a max transmit time (e.g. FCC 400 ms channel dwell limit in 915 MHz band if not hopping). Using smaller packets and a short delay (a few seconds) between them is a safe practice.

3. **Receive and reassemble:** On the receiving side, it's best to have a script continuously listening (`radio rx 0` loop) and reading incoming packets. Each time `radio_rx <data>` appears, parse out the hex payload and convert it back to binary. Append it to an output file in the correct order. You might include a simple sequence number in each packet's payload to keep track of order and detect any missing chunks. For example, reserve the first byte of each payload as a sequence counter.

4. **Acknowledge or handle loss:** LoRa has built-in CRC, so corrupted packets are discarded (you'll either get `radio_rx` with good data or no message at all if CRC failed). However, if a packet is lost due to interference, the sender wouldn't know. For reliability, you can implement an **ACK** scheme at the application layer: after receiving a chunk, the receiver could send back a small ACK packet (e.g. with the sequence number) to confirm. The sender, upon sending each chunk, can switch to RX mode (`radio rx <timeout>`) expecting an ACK. If none arrives, it can retry the chunk. This of course reduces throughput further, but improves reliability – a typical trade-off in designing a protocol.

5. **Complete the transfer:** After the last chunk, the receiver can assemble the file. Compare checksums on both ends to ensure the file transferred correctly.

Note: This is a manual implementation of a file transfer protocol. You might recognize this resembles classic protocols like XModem/YModem (which send packets with checksums and ACKs). You could adapt such protocols to LoRa if desired. But given our low bitrate, a simple custom loop is often easier. For example, a 1 KB file at SF12/125kHz could take tens of seconds to send – plan for the throughput roughly in the range of a few bytes per second at SF12 (at SF7 you'd get faster speeds but less range).

LoRa Communication Protocol Options

You asked about what protocol to use and mentioned using "raw LoRa commands" for now (which we have done via the RN2903's AT-like interface). Here's a summary of your options:

- **Raw LoRa P2P (point-to-point):** This is the approach we used – directly controlling the radio on each end with `radio` commands. It gives you full control and is great for a dedicated link or custom networks. However, **it's up to you to implement any higher-level protocol features** (handshaking, addressing multiple nodes, retries, etc). Our ping and file transfer above are examples of custom, simple protocols on top of LoRa. Raw LoRa is suitable for **point-to-point or simple broadcasts** and

can even be extended to rudimentary mesh by writing forwarding logic in your code, but the module itself does not provide a mesh or routing protocol.

- **LoRaWAN:** This is the standardized MAC protocol *using* LoRa modulation, intended for star-topology networks with a gateway. **LoRaWAN is not used for direct device-to-device links** – instead each device talks to a central gateway. In your scenario (just two devices), LoRaWAN would require one device to act as a gateway or both to connect to an existing gateway, which is not what you want. We explicitly paused the LoRaWAN MAC to do P2P. So, LoRaWAN is *not* the protocol for point-to-point (and the RN2903 cannot become a proper multi-channel gateway) [35]. Stick to raw mode for P2P.

- **Simple LoRa Protocols/Mesh:** If in the future you want a mesh network (multi-hop relaying across several LoRa nodes), you would have to introduce a routing protocol on top of LoRa. There are some open-source projects and research into LoRa mesh networking (for example, Meshtastic and LoRaMesh frameworks, or libraries like RadioHead on Arduino for simple mesh), but these often require custom firmware or specific transceiver setups. With RN2903 modules, you'd likely need to use an external microcontroller or software to handle mesh routing, treating the RN2903 as a modem. For now, **point-to-point is much simpler** – get that working first, then consider scaling up.

- **AT Command vs. Custom Firmware:** Using the RN2903's ASCII command interface (as we did) is the quickest way to get started [36]. An alternative would be to bypass the RN2903's firmware and use a LoRa transceiver module (like HopeRF RFM95/SX1276) directly with a library. Some users find that approach more flexible (since you can get RSSI, control interrupts, etc. which the RN2903's "basic" API doesn't fully expose [37]). However, that requires more hardware work or reflashing the RN2903. Given you have the Microchip modules in hand, using their built-in command set is perfectly fine for justification and initial testing.

In summary, **for your project's point-to-point requirement, raw LoRa mode via the RN2903 AT commands is the recommended approach**. It was even suggested in earlier prompts and by others, as it allows two RN2903 units to talk to each other without any network infrastructure [13]. Just remember that unlike Wi-Fi or Bluetooth, there's no automatic handshake or IP stack – you are effectively creating a custom link layer. This gives flexibility (and that "mesh" idea can be built on it later), but it means you handle the communication logic.

Testing at Range and Justifying LoRa

Once you have the link working at short range on your desk, you can truly **justify the choice of LoRa by testing at long range**. Try moving one device further away (hundreds of meters, then kilometers if possible) and see if Wi-Fi Ad-hoc or Bluetooth can still maintain a connection – they likely won't beyond a few tens of meters. LoRa, on the other hand, should still work over a very long distance if line-of-sight or even partial line-of-sight is available. In real-world tests, even a simple setup with small antennas achieved on the order of **1–2 km in a dense urban environment and 4+ km with some line-of-sight**. With proper antennas and higher transmit power, **suburban ranges of 15 km and urban up to 5 km** are cited for the RN2903 [1]. This is where LoRa shines compared to Wi-Fi Direct or Bluetooth – those technologies cannot reach that far.

Do keep in mind the **legal transmission limits** when testing range. In the 915 MHz ISM band, you may need to limit continuous transmission or use frequency hopping. As one LoRa expert noted, *"read up on the*

legal requirements for the band… how much airtime you can use, how frequently you're allowed to transmit… and use a frequency not used by LoRaWAN in your vicinity to avoid interference." [21] [13] . Practically, for occasional pings and small file transfers at reasonable intervals, you should be fine, but avoid trying to send very large amounts of data continuously over LoRa.

Finally, by demonstrating a successful long-distance ping and file transfer, you will have concrete evidence to **justify using LoRa**: it can connect devices at distances where other wireless methods fail, highlighting the trade-off (range vs. data rate) in your report. Good luck with your testing, and enjoy exploring the capabilities of your LoRa link!

Sources:

- Microchip RN2903 PICtail User Guide – notes on USB interface and antenna requirements [5] [38] .
- *disk91 IoT Blog*: "First steps with Microchip RN2483 LoRa" – example of point-to-point setup and commands (RN2483 is the 868 MHz sibling of RN2903) [2] [39] [40] .
- Microchip RN2903 Command Reference – usage of `mac pause`, `radio tx`/`rx`, and parameter ranges [11] [25] [28] .
- Microchip RN2903 Datasheet – LoRa capabilities and typical range (15 km+ suburban, 5 km urban with proper settings) [1] .
- The Things Network forum – advice on RN2903 point-to-point and legal considerations [13] [21] .

---

[1] MICROCHIP RN2903 Low-Power Long Range LoRa Transceiver Module User Manual
https://manuals.plus/microchip/rn2903-low-power-long-range-lora-transceiver-module-manual

[2] [7] [8] [9] [10] [15] [17] [19] [23] [24] [27] [29] [32] [33] [39] [40] First step in LoRa land – microchip RN2483 test | disk91.com – the IoT blog
https://www.disk91.com/2015/technology/networks/first-step-in-lora-land-microchip-rn2483-test/

[3] [36] Build Long-Range IoT Networks with Microchip's RN2903 LoRa Transceiver Module, Now at Mouser
https://eu.mouser.com/publicrelations_microchip_rn2903_lora_transceiver_2016final/?srsltid=AfmBOoppEqiQy7SKNAQD8cLMWpw-TUvfruZYgBjPeiweHmev7u7ICI2m

[4] [5] www.microchip.com
https://www.microchip.com/en-us/development-tool/rn-2903-pictail

[6] [38] RN2903 LoRa Technology PICtail/PICtail Plus Daughter Board User's Guide
https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/UserGuides/50002424A.pdf

[11] [12] [14] [16] [18] [20] [25] [26] [28] [30] [31] [34] dsPICDEM™ MCHV-2 Development Board User's Guide
https://ww1.microchip.com/downloads/en/DeviceDoc/RN2903%20LoRa%20Technology%20Module%20Command%20Reference%20User%20Guide-DS40001811B.pdf

[13] [21] [22] [35] [37] Setting up the Rpi 3B+ with the Microchip RN2903A click from MicroE - How to get started - The Things Network
https://www.thethingsnetwork.org/forum/t/setting-up-the-rpi-3b-with-the-microchip-rn2903a-click-from-microe/16554