# Shells on Constrained Devices

## Using CoAP to explore new UX paradigms for embedded shells

**Bennet Blischke 11.07.2024**

# What is a shell?

- The interface between User and Operating System
  - Command Line Interface (CLI)
  - Graphical User Interface (GUI)

- They form the outer ***shell*** of the computing system

image: Flaticon.com

# What is a shell?

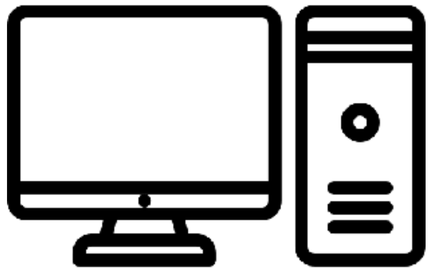
- The interface between User and Operating System
  - **Command Line Interface (CLI)**
  - Graphical User Interface (GUI)

- They form the outer ***shell*** of the computing system

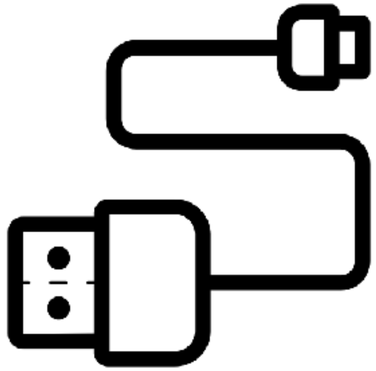
image: Flaticon.com

# How are Shells used on Constrained Devices?

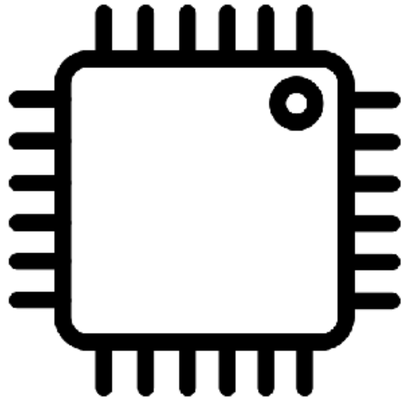
## A RIOT OS perspective



Programmer's computer



Typical Connection: UART over USB



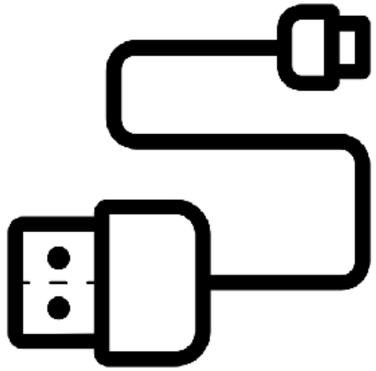
Embedded device

Exchange of ASCII characters

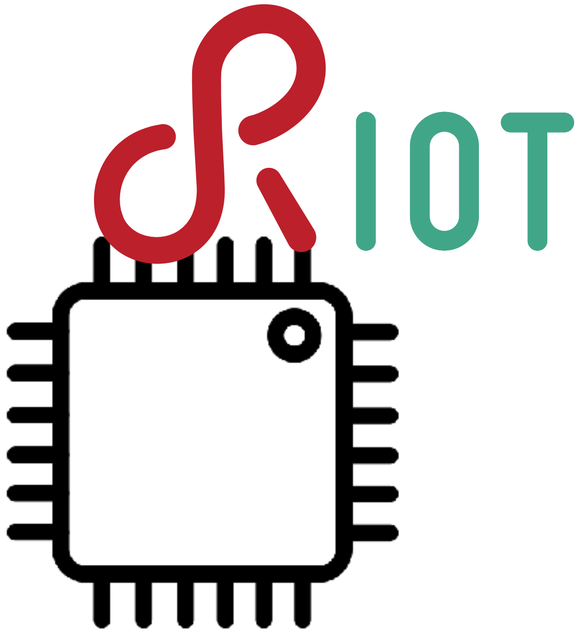
# How are Shells used on Constrained Devices?

## A RIOT OS perspective

Programmer's computer



Typical Connection: UART over USB




Embedded device

Exchange of ASCII characters

# How are Shells used on Constrained Devices?

## A RIOT OS perspective

```
teufelchen@teufelchen-OptiPlex-7070:~/Programming/RIOT/examples/default$ make term
```

# How are Shells used on Constrained Devices?

## A RIOT OS perspective

```
teufelchen@teufelchen-OptiPlex-7070:~/Programming/RIOT/examples/default$ make term
/home/teufelchen/Programming/RIOT/dist/tools/pyterm/pyterm -p "/dev/ttyACM0" -b "115200"
Twisted not available, please install it if you want to use pyterm's JSON capabilities
2024-07-09 11:05:16,417 # Connect to serial port /dev/ttyACM0
Welcome to pyterm!
Type '/exit' to exit.
```

# How are Shells used on Constrained Devices?

## A RIOT OS perspective

```
teufelchen@teufelchen-OptiPlex-7070:~/Programming/RIOT/examples/default$ make term
/home/teufelchen/Programming/RIOT/dist/tools/pyterm/pyterm -p "/dev/ttyACM0" -b "115200"
Twisted not available, please install it if you want to use pyterm's JSON capabilities
2024-07-09 11:05:16,417 # Connect to serial port /dev/ttyACM0
Welcome to pyterm!
Type '/exit' to exit.
help
2024-07-09 11:05:20,257 # help
2024-07-09 11:05:20,260 # Command              Description
2024-07-09 11:05:20,263 # ---------------------------------------
2024-07-09 11:05:20,268 # ifconfig             Configure network interfaces
2024-07-09 11:05:20,272 # nib                  Configure neighbor information base
2024-07-09 11:05:20,277 # pm                   interact with layered PM subsystem
2024-07-09 11:05:20,283 # ps_regular           Prints information about running threads.
2024-07-09 11:05:20,286 # reboot               Reboot the node
2024-07-09 11:05:20,292 # saul                 interact with sensors and actuators using SAUL
2024-07-09 11:05:20,298 # txtsnd               Sends a custom string as is over the link layer
2024-07-09 11:05:20,302 # version              Prints current RIOT_VERSION
>
```

# How are Shells used on Constrained Devices?

## A RIOT OS perspective

```
teufelchen@teufelchen-OptiPlex-7070:~/Programming/RIOT/examples/default$ make term
/home/teufelchen/Programming/RIOT/dist/tools/pyterm/pyterm -p "/dev/ttyACM0" -b "115200"
Twisted not available, please install it if you want to use pyterm's JSON capabilities
2024-07-09 11:05:16,417 # Connect to serial port /dev/ttyACM0
Welcome to pyterm!
Type '/exit' to exit.
help
2024-07-09 11:05:20,257 # help
2024-07-09 11:05:20,260 # Command              Description
2024-07-09 11:05:20,263 # ---------------------------------------
2024-07-09 11:05:20,268 # ifconfig             Configure network interfaces
2024-07-09 11:05:20,272 # nib                  Configure neighbor information base
2024-07-09 11:05:20,277 # pm                   interact with layered PM subsystem
2024-07-09 11:05:20,283 # ps_regular           Prints information about running threads.
2024-07-09 11:05:20,286 # reboot               Reboot the node
2024-07-09 11:05:20,292 # saul                 interact with sensors and actuators using SAUL
2024-07-09 11:05:20,298 # txtsnd              Sends a custom string as is over the link layer
2024-07-09 11:05:20,302 # version            Prints current RIOT_VERSION
> ps_regular
2024-07-09 11:05:25,739 # ps_regular
2024-07-09 11:05:25,746 # name                 | stack  ( used) ( free) |base addr |current
2024-07-09 11:05:25,752 # isr_stack            |    512 (  224) (  288) |0x20000000|0x20000154
2024-07-09 11:05:25,758 # main                 |   1536 (  648) (  888) |0x20000434|0x200007cc
2024-07-09 11:05:25,764 # pktdump             |   1472 (  176) ( 1296) |0x200019c4|0x20001ed4
2024-07-09 11:05:25,770 # 6lo                  |    960 (  216) (  744) |0x20001fcc|0x200022cc
2024-07-09 11:05:25,776 # ipv6                 |    960 (  440) (  520) |0x20000a9c|0x20000cec
2024-07-09 11:05:25,782 # udp                  |    448 (  196) (  252) |0x200023d0|0x200024cc
2024-07-09 11:05:25,788 # nrf802154            |    896 (  292) (  604) |0x200012e4|0x200015a4
>
```

# How are Shells used on Constrained Devices?

## A RIOT OS perspective

```
teufelchen@teufelchen-OptiPlex-7070:~/Programming/RIOT/examples/default$ make term
/home/teufelchen/Programming/RIOT/dist/tools/pyterm/pyterm -p "/dev/ttyACM0" -b "115200"
Twisted not available, please install it if you want to use pyterm's JSON capabilities
2024-07-09 11:05:16,417 # Connect to serial port /dev/ttyACM0
Welcome to pyterm!
Type '/exit' to exit.
help
2024-07-09 11:05:20,257 # help
2024-07-09 11:05:20,260 # Command              Description
2024-07-09 11:05:20,263 # ---------------------------------------
2024-07-09 11:05:20,268 # ifconfig             Configure network interfaces
2024-07-09 11:05:20,272 # nib                  Configure neighbor information base
2024-07-09 11:05:20,277 # pm                   interact with layered PM subsystem
2024-07-09 11:05:20,283 # ps_regular           Prints information about running threads.
2024-07-09 11:05:20,286 # reboot               Reboot the node
2024-07-09 11:05:20,292 # saul                 interact with sensors and actuators using SAUL
2024-07-09 11:05:20,298 # txtsnd              Sends a custom string as is over the link layer
2024-07-09 11:05:20,302 # version            Prints current RIOT_VERSION
> ps_regular
2024-07-09 11:05:25,739 # ps_regular
2024-07-09 11:05:25,746 # name                 | stack  ( used) ( free) |base addr |current
2024-07-09 11:05:25,752 # isr_stack            |    512 (  224) (  288) |0x20000000|0x20000154
2024-07-09 11:05:25,758 # main                 |   1536 (  648) (  888) |0x20000434|0x200007cc
2024-07-09 11:05:25,764 # pktdump             |   1472 (  176) ( 1296) |0x200019c4|0x20001ed4
2024-07-09 11:05:25,770 # 6lo                  |    960 (  216) (  744) |0x20001fcc|0x200022cc
2024-07-09 11:05:25,776 # ipv6                 |    960 (  440) (  520) |0x20000a9c|0x20000cec
2024-07-09 11:05:25,782 # udp                  |    448 (  196) (  252) |0x200023d0|0x200024cc
2024-07-09 11:05:25,788 # nrf802154            |    896 (  292) (  604) |0x200012e4|0x200015a4
> reboot
2024-07-09 11:05:35,533 # reboot
2024-07-09 11:05:35,542 # main(): This is RIOT! (Version: 2024.07-devel-135-g46924-feat/slipconf)
2024-07-09 11:05:35,543 # Welcome to RIOT!
>
```

# Shortcomings

- ROM / RAM overhead due to string-parsing on the constrained device
- It’s complicated to exchange binary data via an ASCII stream
- No correlation between received text and origin
    - First thread starts to write
    - Another thread interrupts & writes its own message
    - First thread continues and finishes its message
    - The re out of order sult is hard to read and

# Shortcomings

- Long running and highly asynchronous tasks block the shell
- Difficulties in unit-/automated testing of shell components
- Signalling of success and failure of commands

# The Issue with String Parsing

## ROM / RAM overhead due to string parsing on the constrained device

- Strings require a lot of ROM, one byte per character
  - TOP That just was 53 bytes!
- Even worse, when formatting strings. Example:
  - printf(“Year %d”, year);
  - 7 bytes for the format string
  - 9 bytes for the result “Year 2024”

# The Issue with String Parsing

## ROM / RAM overhead due to string parsing on the constrained device

- We do a lot with strings in our shells
- A command takes strings as parameter: ifconfig(int argc, char *argv[])
- Now, parse “ifconfig 7 set channel 0x21”
- A command outputs its result by printing formatted strings
    - Try printing a table of running threads

# How do We Improve?

## Format Strings

- Deferred Formatting:
  1. Assign all format strings a short ID
  2. Only compile the ID into the firmware
  3. When printing, send the ID + format values as it
  4. On the host, look up corresponding strings for received IDs
  5. Format and print on the host side

For an example see:

https://defmt.ferrous-systems.com/

# How do We Improve?

## Correlating output to commands

- We need a lightweight **A**pplication **P**rotocol
- Suitable for **Co**nstrained devices
- Allows to issue a command with optional parameters
- Provides request & response matching
- Can handle errors:
    - On the host side e.g. “invalid parameter for command”
    - On the IoT side e.g. “failed to enable SPI”

# How do We Improve?

## Correlating output to commands

- We need a lightweight **A**pplication **P**roto
- Suitable for **Co**nstrained devices
- Allows to issue a comma meters
- Provides request &
- Can handle er
  - On the host e.g. “invalid parameter for command”
  - On the IoT side e.g. “failed to enable SPI”

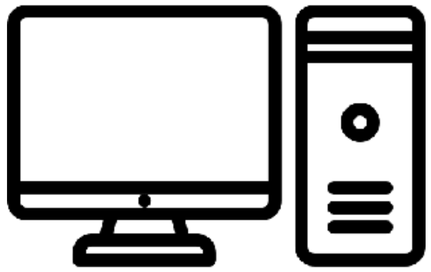
That’s CoAP!

# Even More CoAP Benefits!

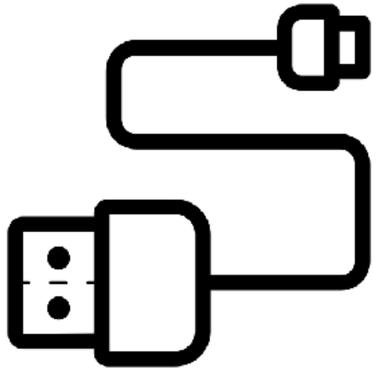
- Completely asynchron: Request now, get the response later!
- With CoAP Observe, we can have timer, events and interrupts in our shell
- Big binary payload? CoAP block-wise transfer saves the day
- Robust ecosystem due to related standards:
    - Discover shell commands via /.well-known/core
    - Describe commands using CoRE Link Format

# CoAP transports via UDP…

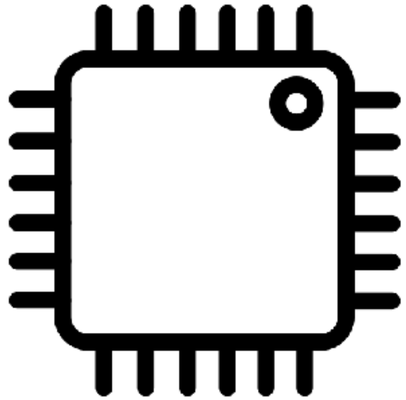
## … but we don’t have UDP, not even IPv6!



Programmer’s computer



Typical Connection: UART over USB



Embedded device

# SLIP

## Serial Line IP

- Very old RFC from 1988
- Short framing and escaping protocol
- Easily send IP frames over serial / UART

# SLIP

## Serial Line IP

- Very old RFC from 1988
- Short framing and escaping protocol
- Easily send IP frames over serial / UART

Can we speak CoAP over UDP over IP via SLIP?

# SLIP

## Serial Line IP

- Very old RFC from 1988
- Short framing and escaping protocol
- Easily send IP frames over serial / UART

Can we speak CoAP over UDP over IP via SLIP?

Yes!

But that’s a lot of overhead, we just want CoAP

# SLIPMUX

## Using an UART interface for diagnostics, configuration, and packet transfer

- New draft from 2019
- Builds on top of SLIP, but adds multiplexing for diagnostic and configuration

# SLIPMUX

## Using an UART interface for diagnostics, configuration, and packet transfer

- New draft from 2019
- Builds on top of SLIP but adds multiplexing for diagnostic and configuration
- **Diagnostic** messages are just UTF-8 **strings**
  - Neat! We can use that for backwards compatibility to the existing shell!

# SLIPMUX

## Using an UART interface for diagnostics, configuration, and packet transfer

- New draft from 2019
- Builds on top of SLIP but adds multiplexing for diagnostic and configuration
- **Diagnostic** messages are just UTF-8 **strings**
  - Neat! We can use that for backwards compatibility to the existing shell!
- **Configuration** messages are **CoAP packages**!
  - This also solves the issue of finding the IP addresses of both participants

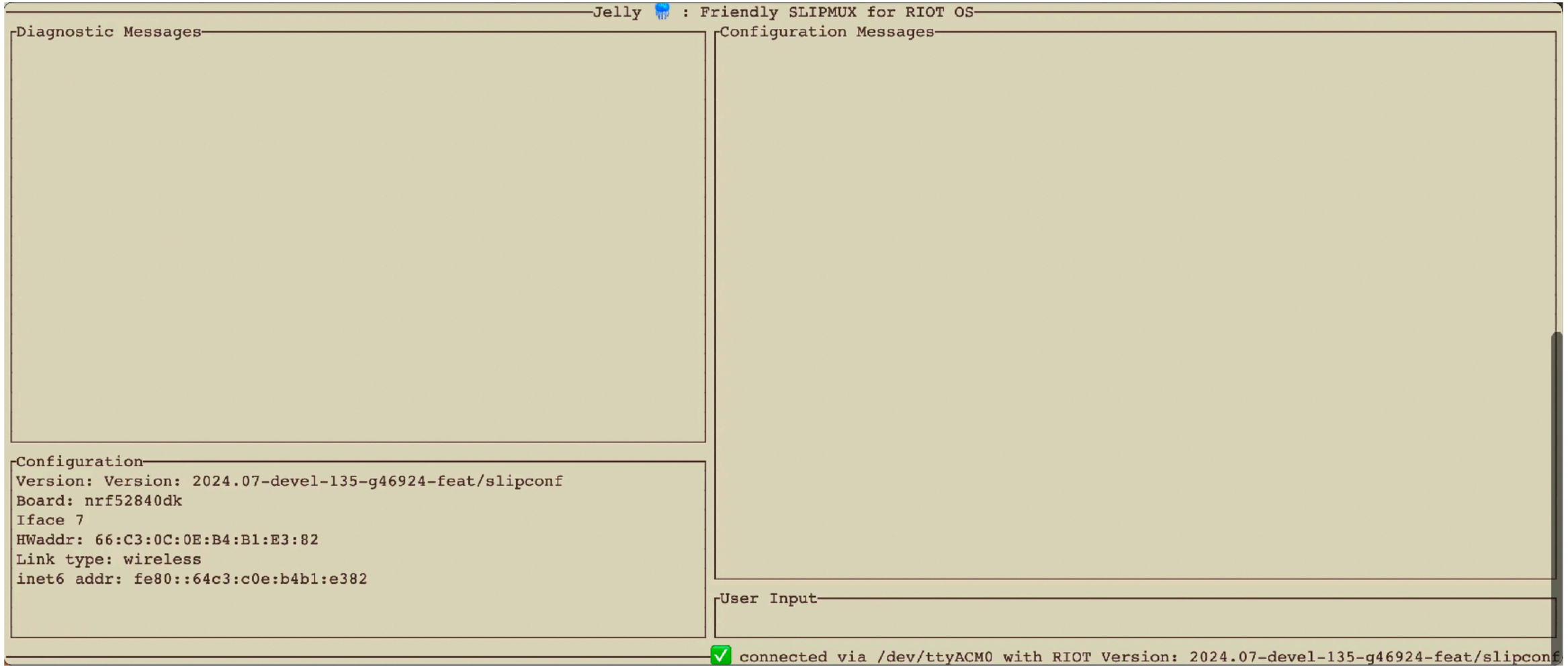
# SLIPMUX

## Using an UART interface for diagnostics, configuration, and packet transfer

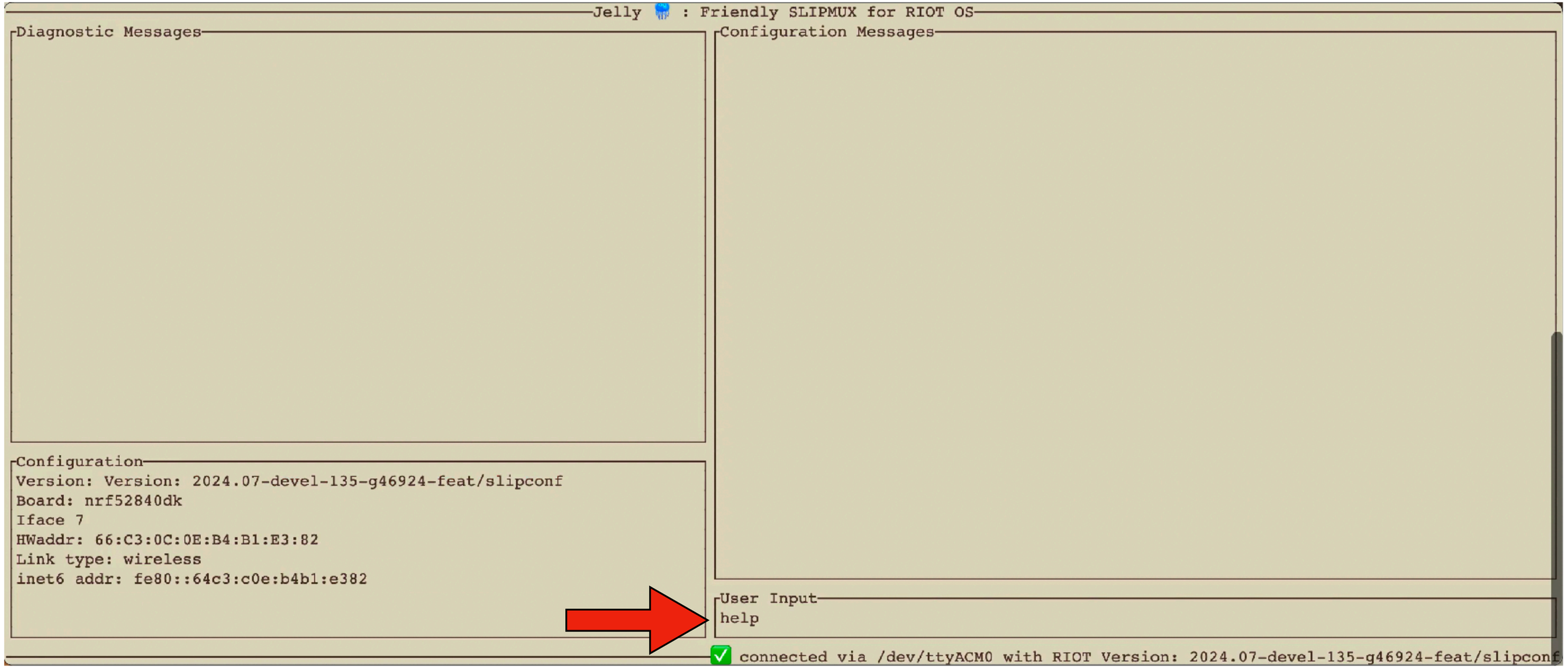
- New draft from 2019
- Builds on top of SLIP but adds multiplexing for diagnostic and configuration
- **Diagnostic** messages are just UTF-8 **strings**
  - Neat! We can use that for backwards compatibility to the existing shell!
- **Configuration** messages are **CoAP packages**!
  - This also solves the issue of finding the IP addresses of both participants
- And if we want, we can still exchange IP frames

# Let’s prototype!

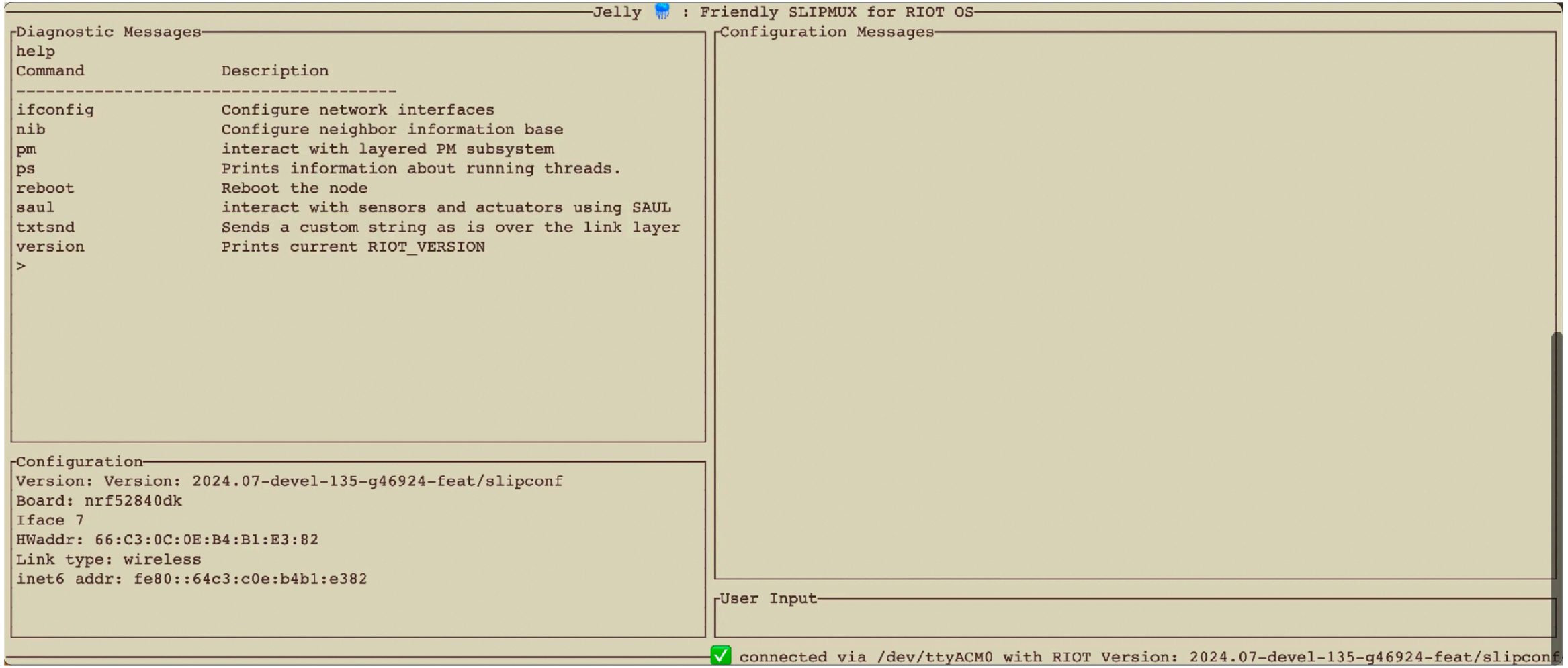
## What does a SLIPMUX client look like?



# Backwards compatible: help

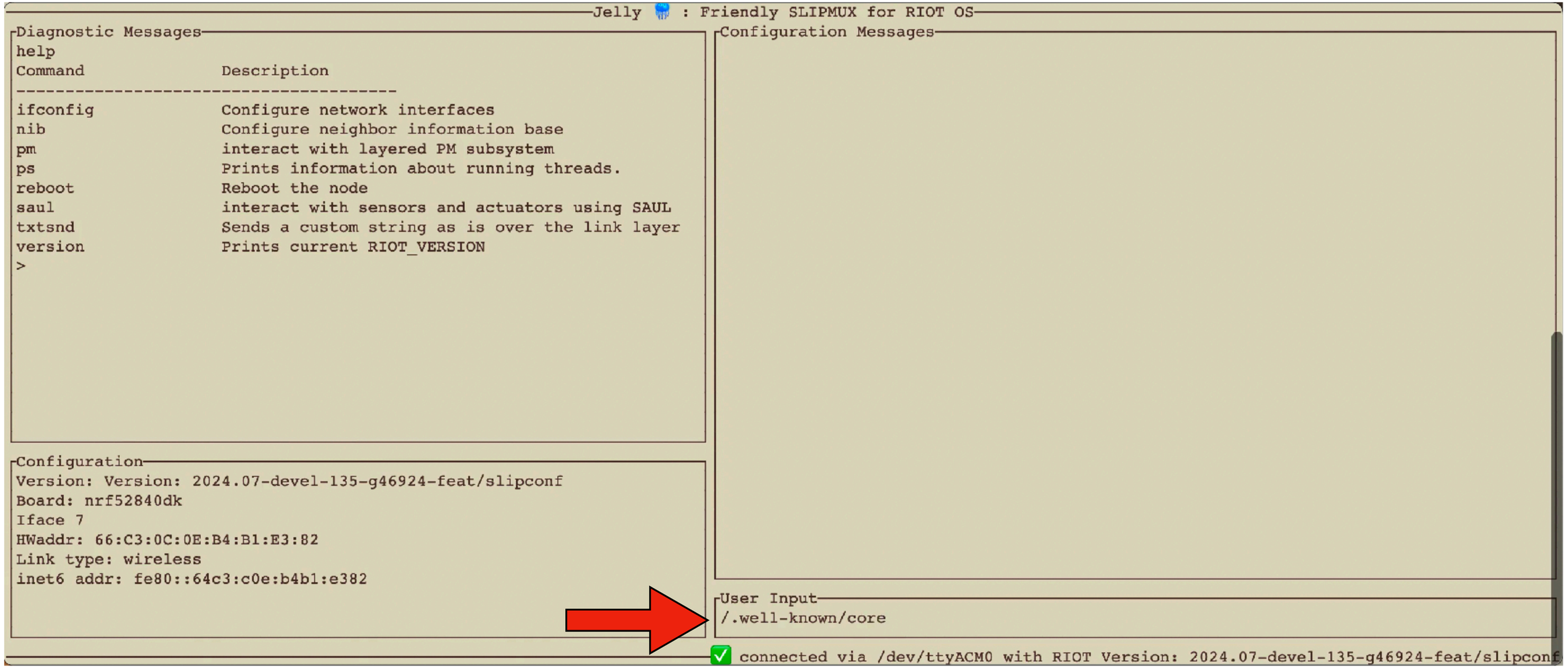


# Backwards compatible: help



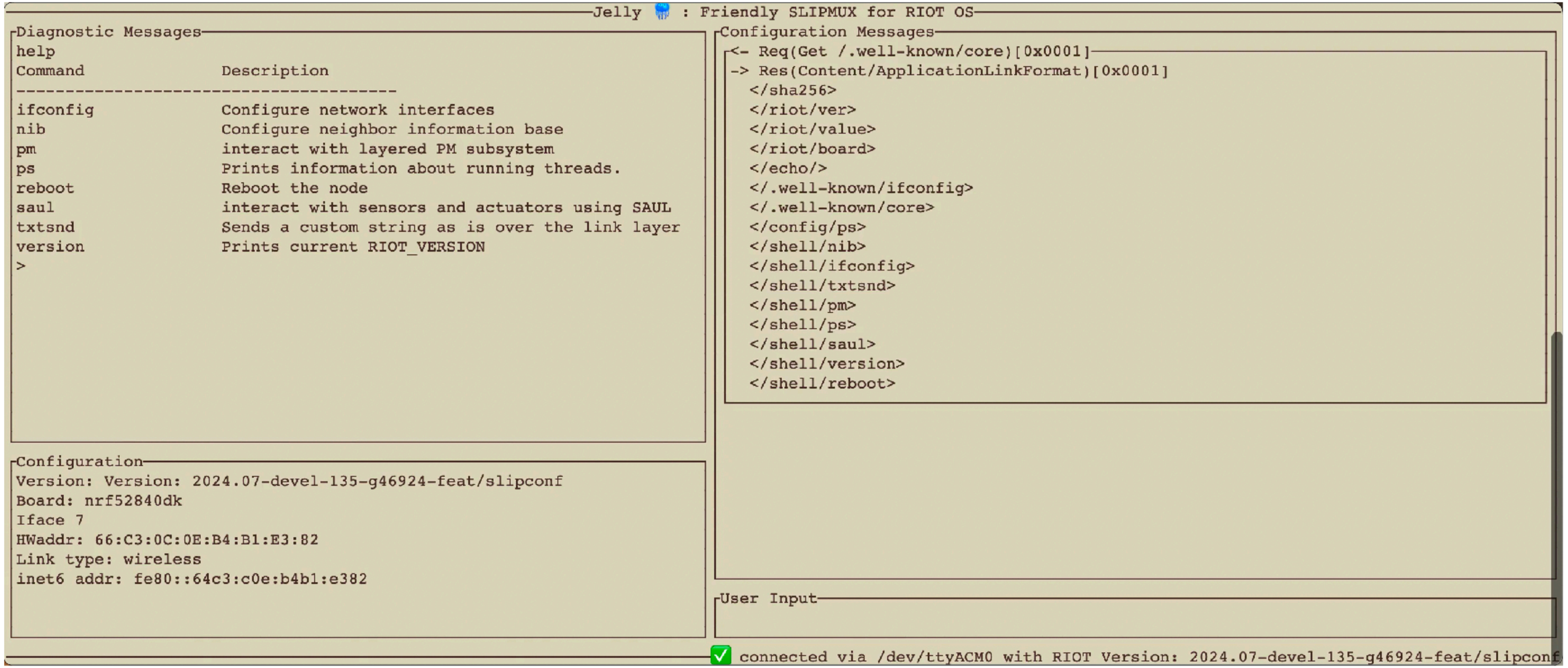
# New: Direct CoAP Requests

## GET /.well-known/core

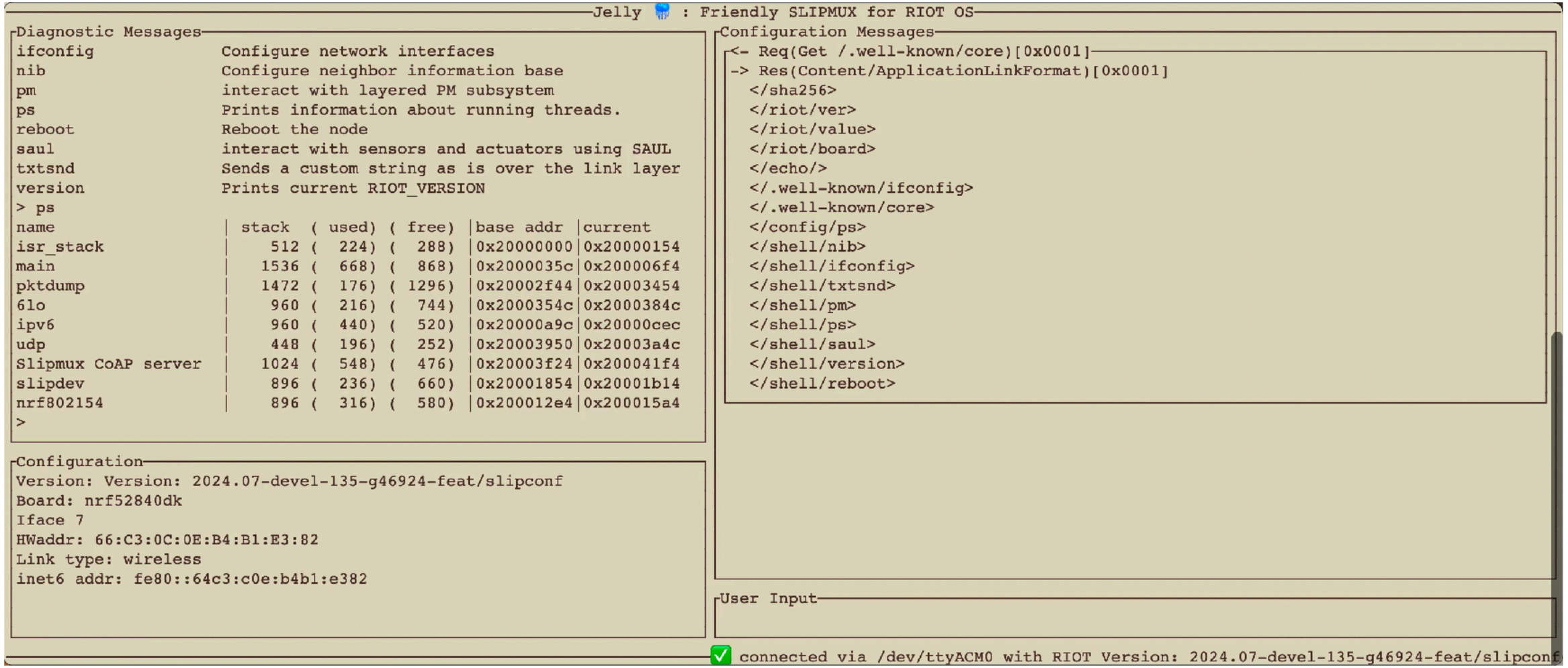


# New: Direct CoAP Requests

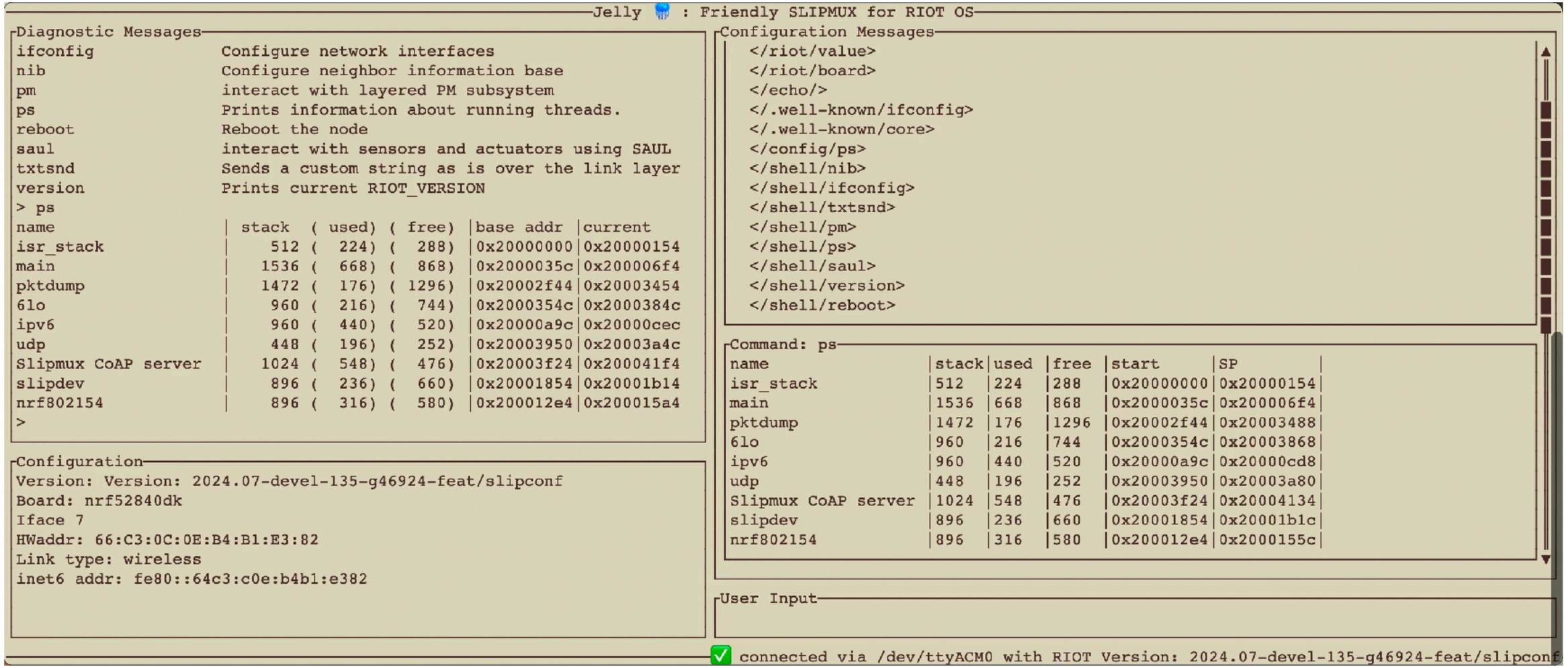
## GET /.well-known/core



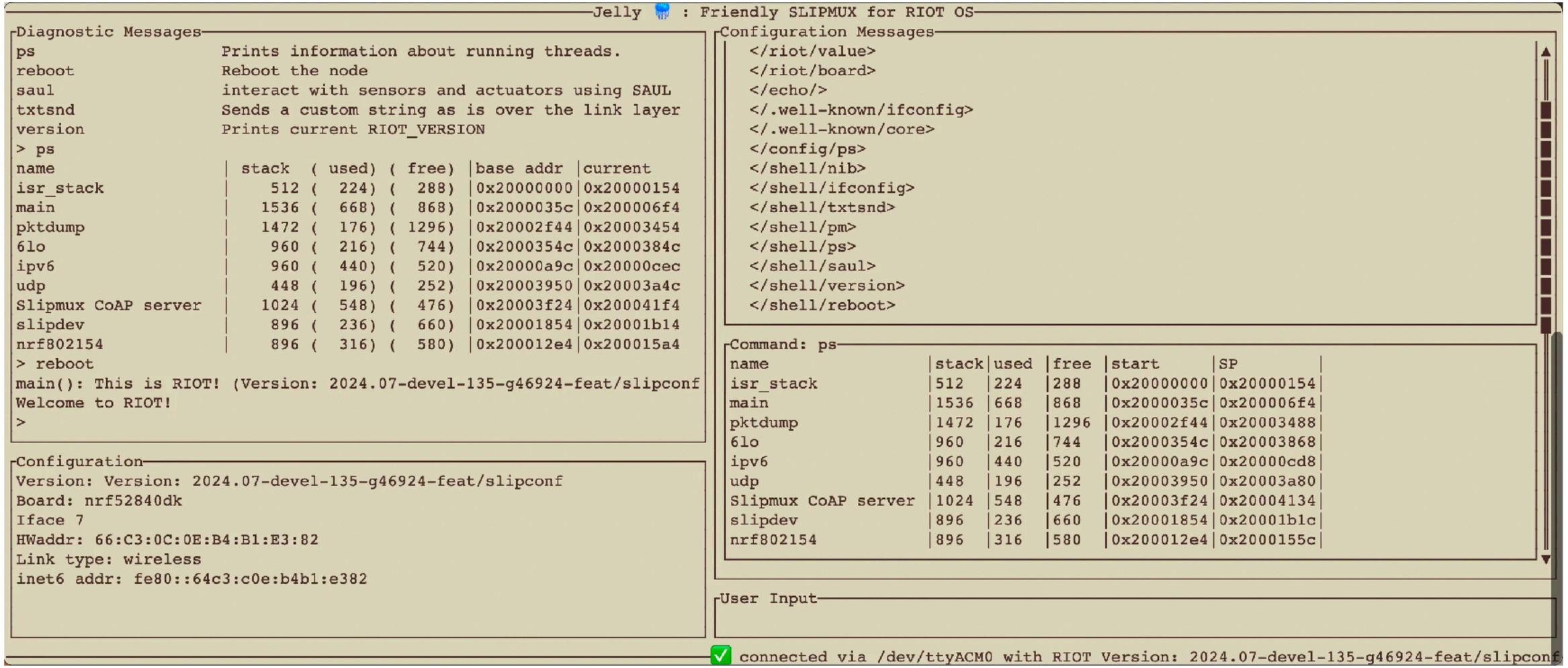
# Backwards Compatible: ps



# Commands via CoAP: ps

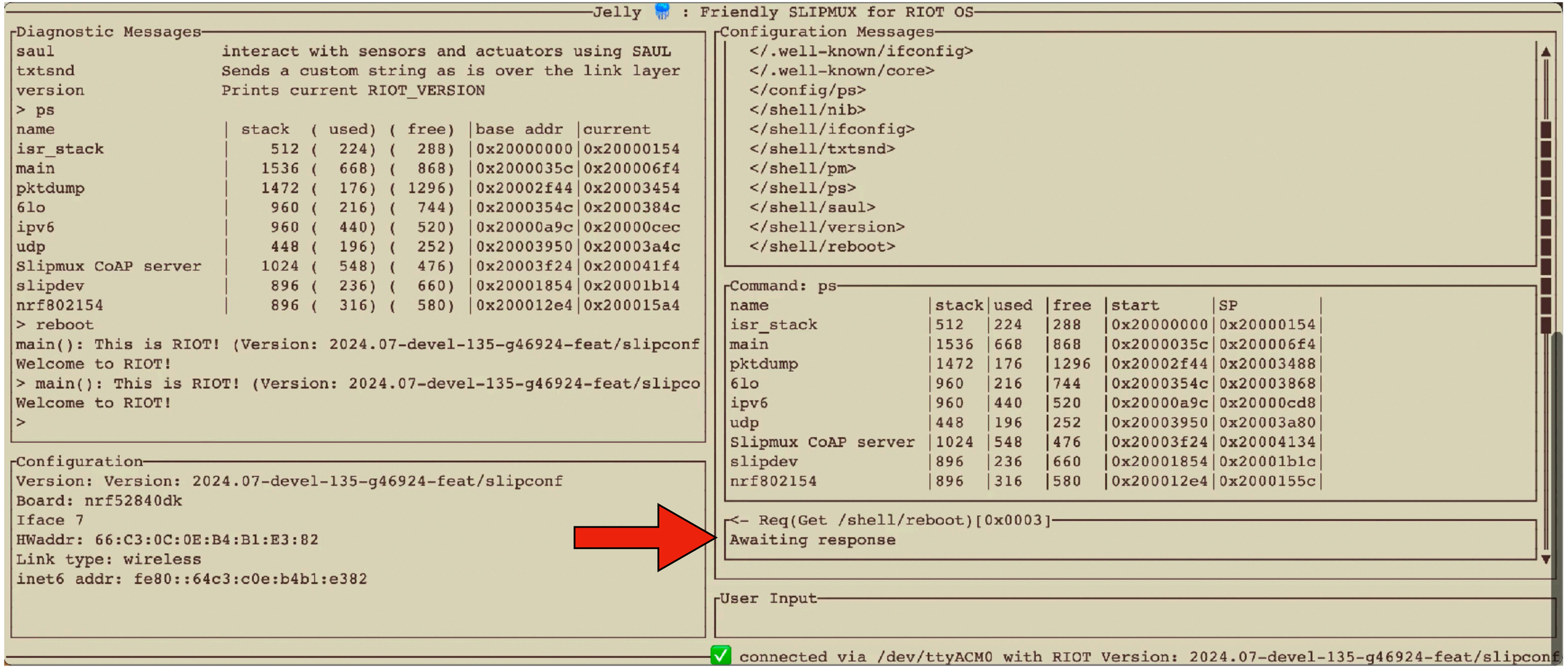


# Backwards Compatible: reboot



# Commands via CoAP: reboot

## Async!



# That was more than just a Shell!

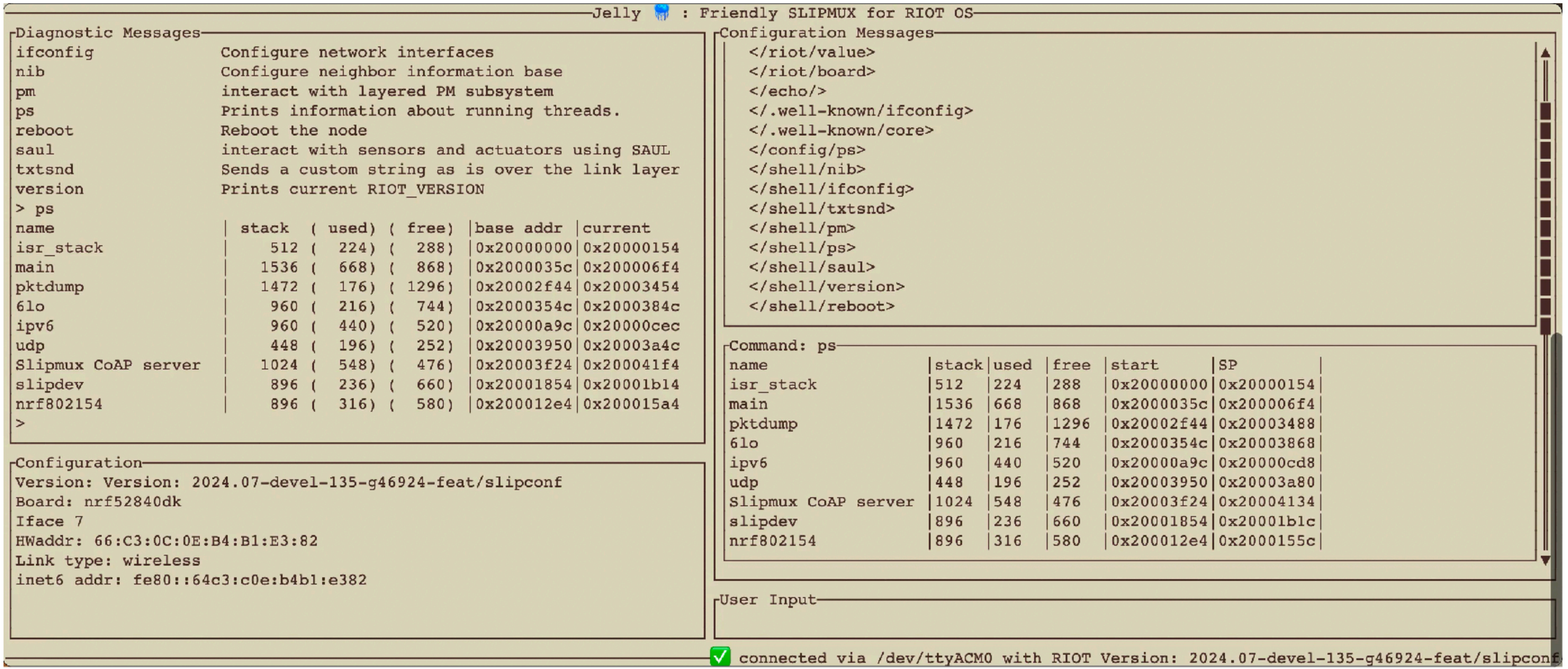
- Because CoAP is not build for human consumption but for machines, we can do cool things now:
    - Auto detect and auto complete available shell commands
    - Automagically fetch and display useful debugging informations such as version and IP addresses of the constrained device
    - Digest binary payloads: present them human readable, save to disk, ..
    - Chain multiple shell commands, passing their I/O into each other

# Conclusion & Outlook

- It is feasible to use CoAP as a basis for a modern shell
- Greatly improved usability
- But there is so much more todo!
    - Structured parameter and return values of commands, e.g. CBOR
    - Interoperability between “new” and “old” shell
    - Security aspects: Opening the shell to the network
    - ROM increase due to CoAP vs. decrease due to reduced string parsing

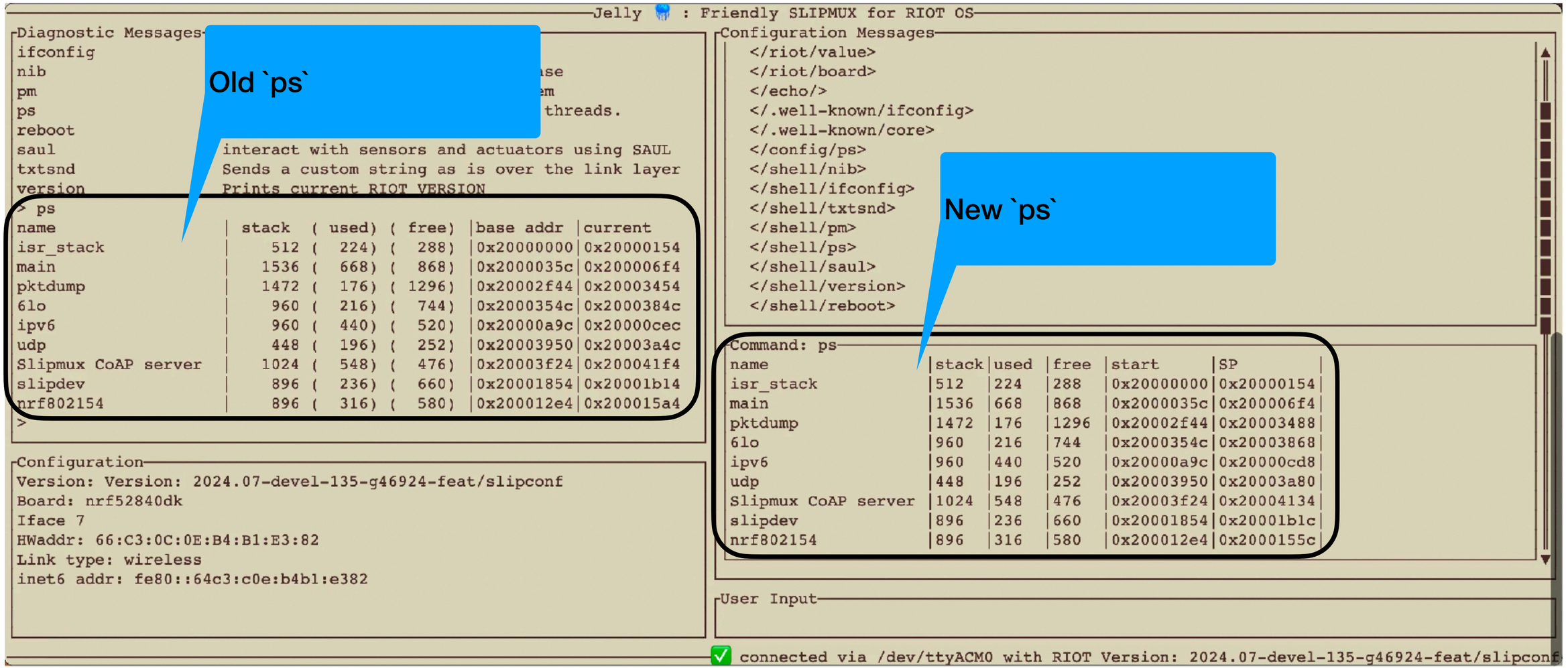
# Bonus Slide: Early Overhead Estimations

## Comparing the old `ps` vs. the new CoAP `ps`



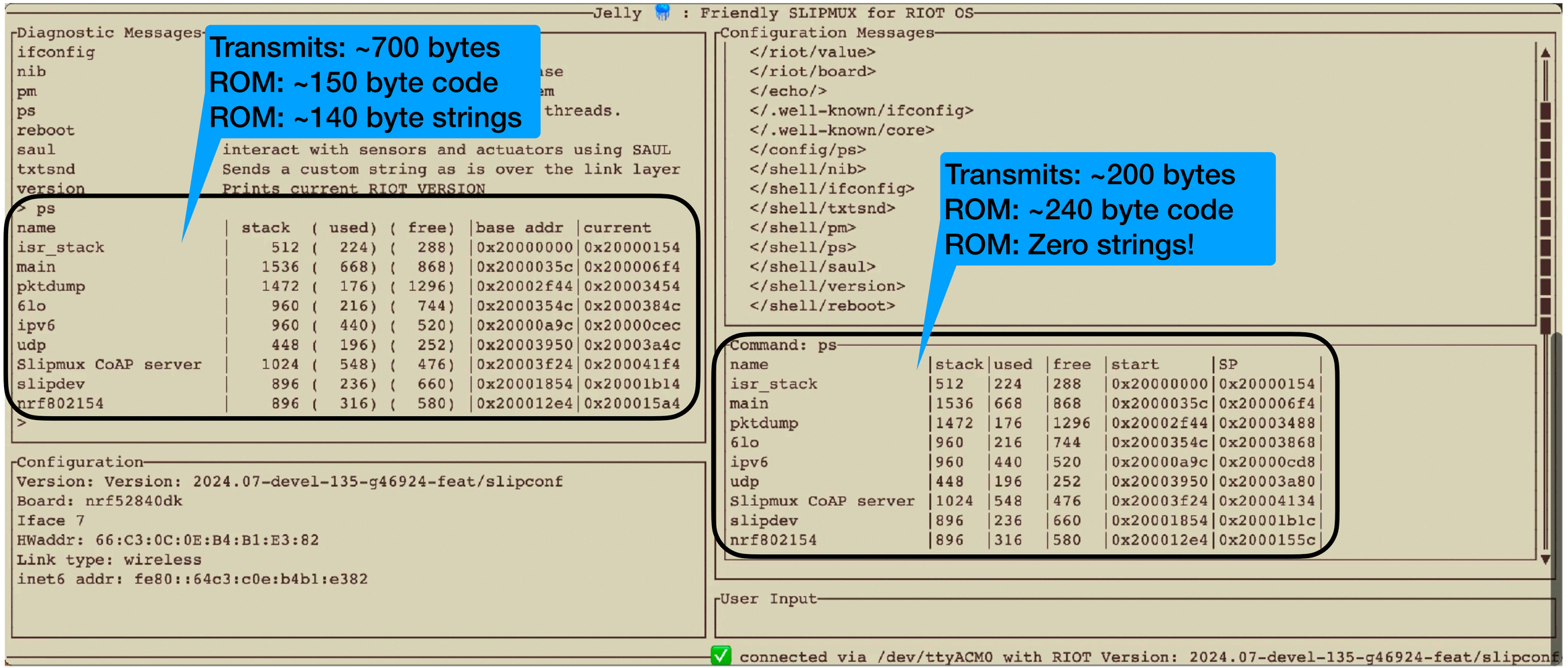
# Bonus Slide: Early Overhead Estimations

## Comparing the old `ps` vs. the new CoAP `ps`



# Bonus Slide: Early Overhead Estimations

## Comparing the old `ps` vs. the new CoAP `ps`



# Thank you for having me!