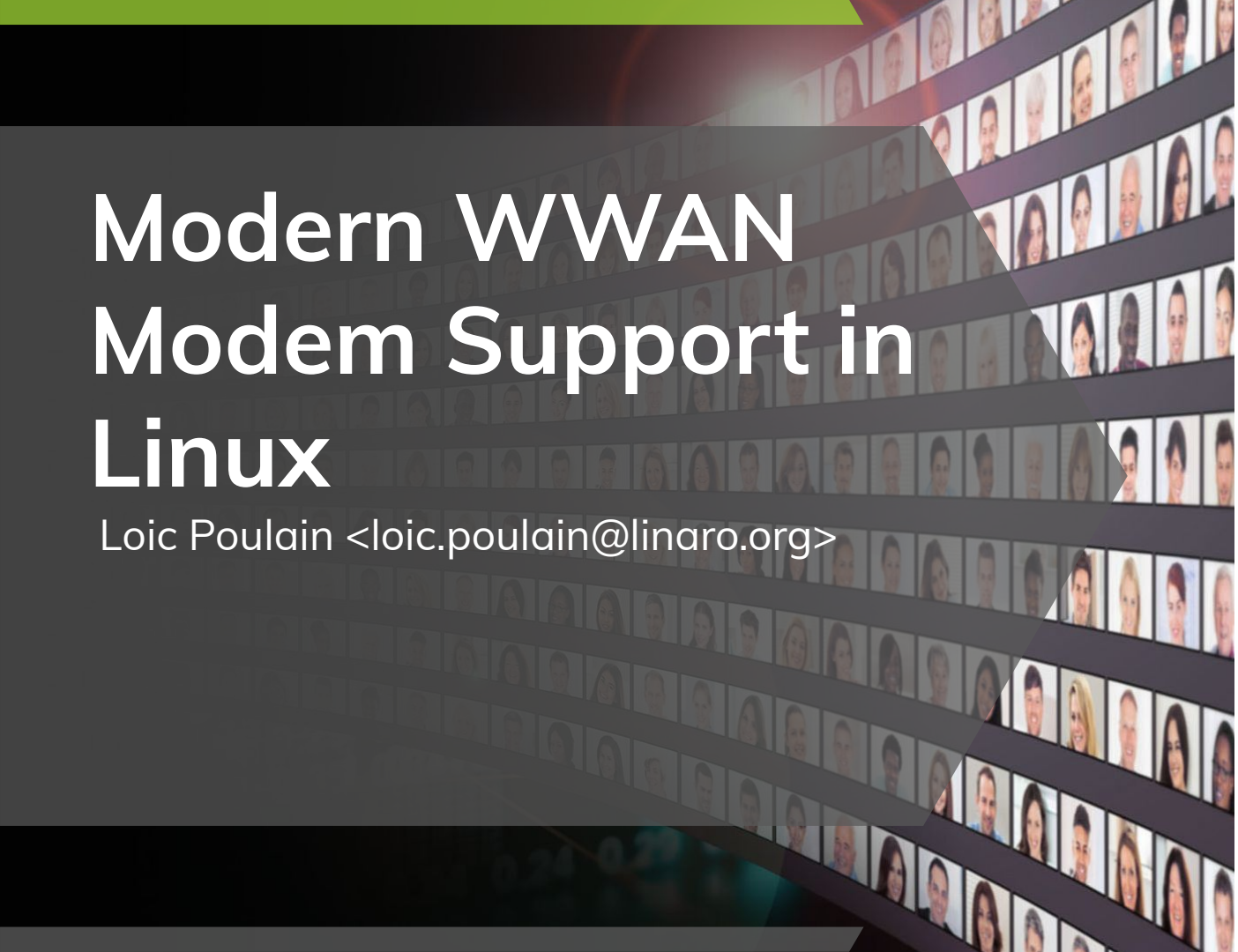
Modern WWAN Modem Support in Linux

Loic Poulain <loic.poulain@linaro.org>



Linaro Connect
VIRTUAL • SPRING 2021

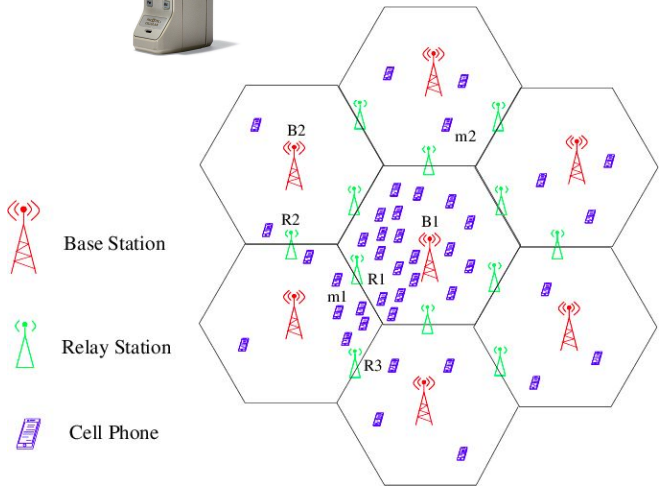
Introduction

A bit of history

Cellular network initially designed in 1980s to transport **voice** to and from **mobile phones** over the air.

The cellular network is distributed over land areas called **‘cells’**, each served by at least one fixed-location transceiver. When joined together, the cells provide radio coverage over a wide geographic area.

Over the years, and under the umbrella of the **3gpp** consortium, cellular networks evolved to offer much more than voice...



From voice to multimedia and industry

1G - 1979 - *NMT, AMPS, TACS*

Analog voice - Voice is modulated to higher frequencies, relying on **circuit switched** technology (CS).

2G - 1991 - *GSM, CDMA, +GPRS, +CDMA2000 +EDGE* *[max 384kbps]*

Digital voice - More users, encrypted data, roaming... introduction of **Data services**, starting with SMS. GPRS (2.5G) brings **packet switched** technology (PS), used for multimedia messages (MMS), Tele-control, opening the doors to the world of the Internet (IP data) => **WWAN**

3G - 2002 - *UMTS, EV-DO, +HSPA+* *[max 42mbps]*

Enhanced voice and introduction of video calls (CS). Improved PS data communication for internet usages such as web browsing, tele-services, remote machine access.

4G - 2009 - *LTE* *[max 1gbps]*

PS only communication (no CS), with **'all-IP' networks**. Allowing HD multimedia streaming, internet gateways/routers. Voice (phone calls) using VoLTE or a *Circuit-switched fallback*.

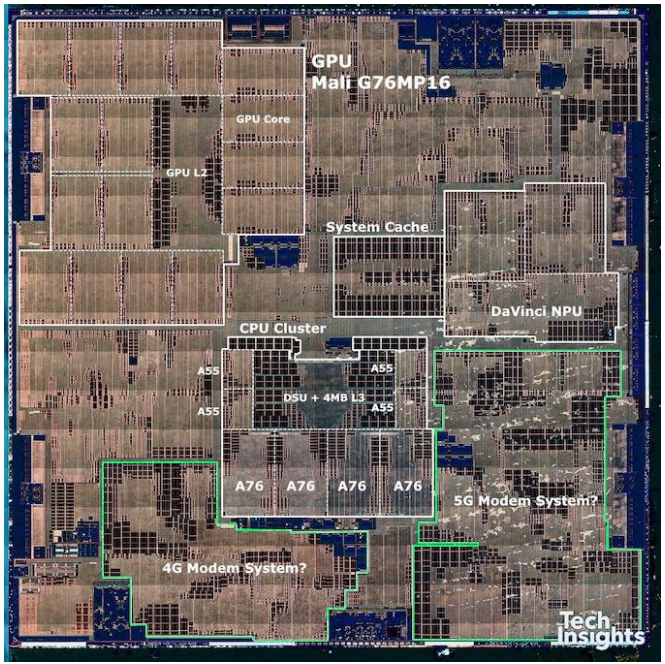
5G - 2019 - *5G NR* *[max 10gbps]*

Continuation of 4G, with higher throughputs and lower latencies, allowing massive usage, and enabling/enhancing edge computing, wireless backhaul, autonomous machines....

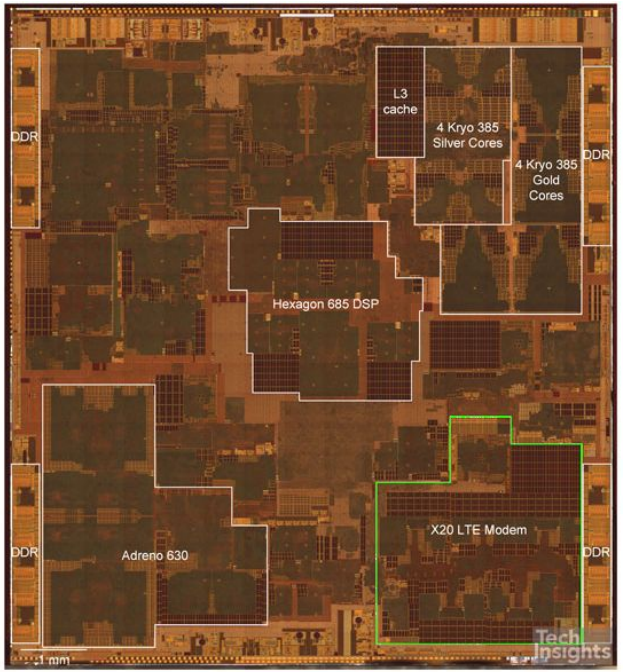
Modems are complex

In modern mobile system-on-a-chips A substantial area of the die is dedicated to the modem IP(s).

Kirin 990



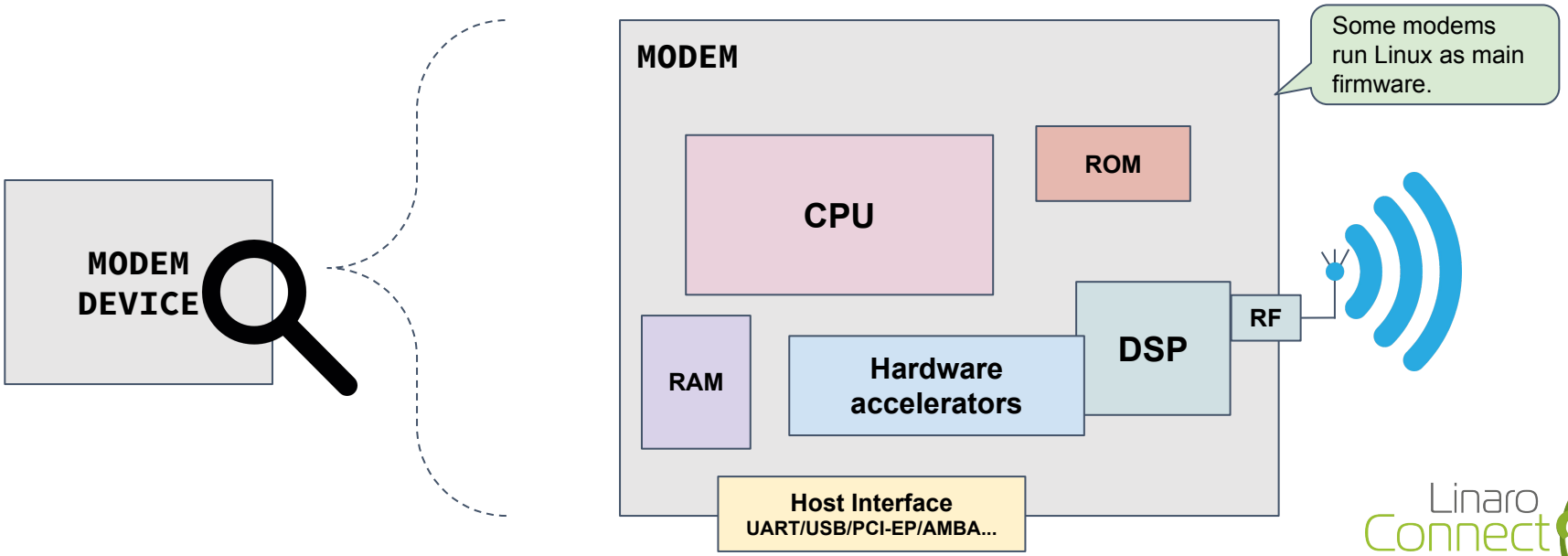
Qualcomm Snapdragon 845



Source: https://www.techinsights.com

Modems are complex

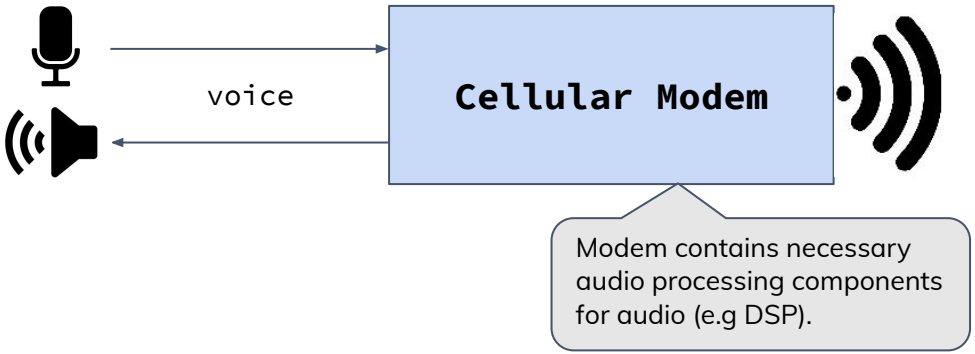
Most of the cellular protocols complexity (as specified by 3gpp) is hidden to the host. The modems integrate various components to implement and abstract this complexity. Some modems offer additional features such as GPS/GNSS, sensors...



The WWAN & Linux story

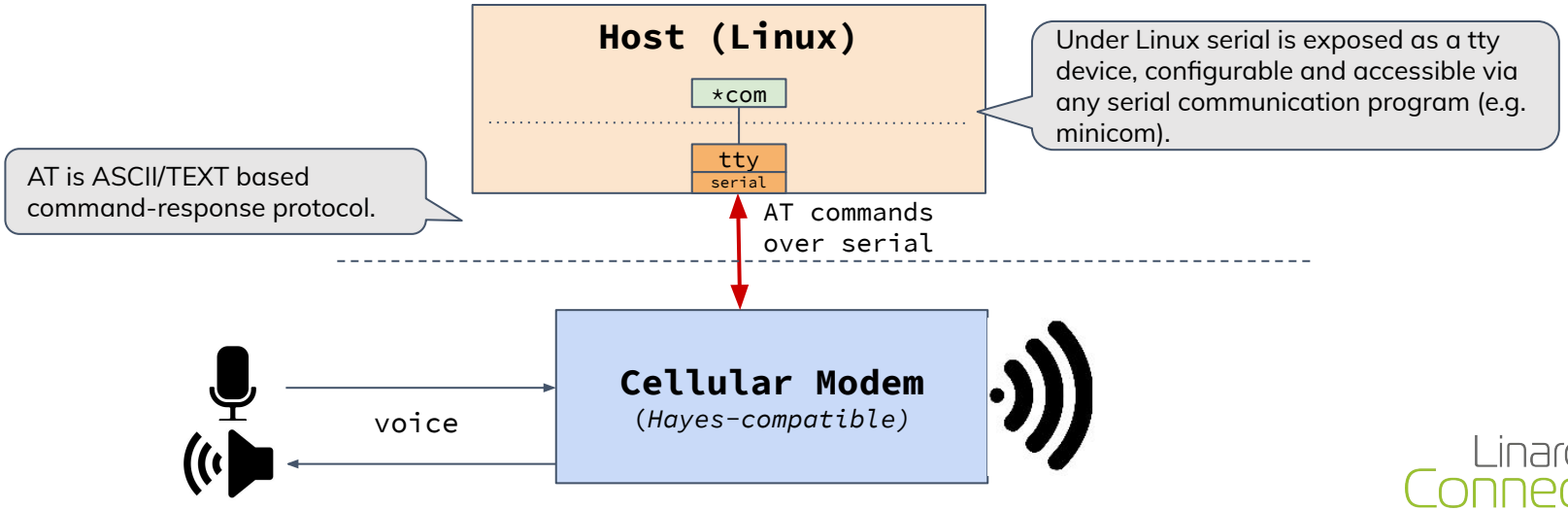
Transporting voice

Initially designed for phone calls, a cellular modems job was to transport voice. Voice path was usually directly interfaced with voice hardware elements (mic, speakers...).



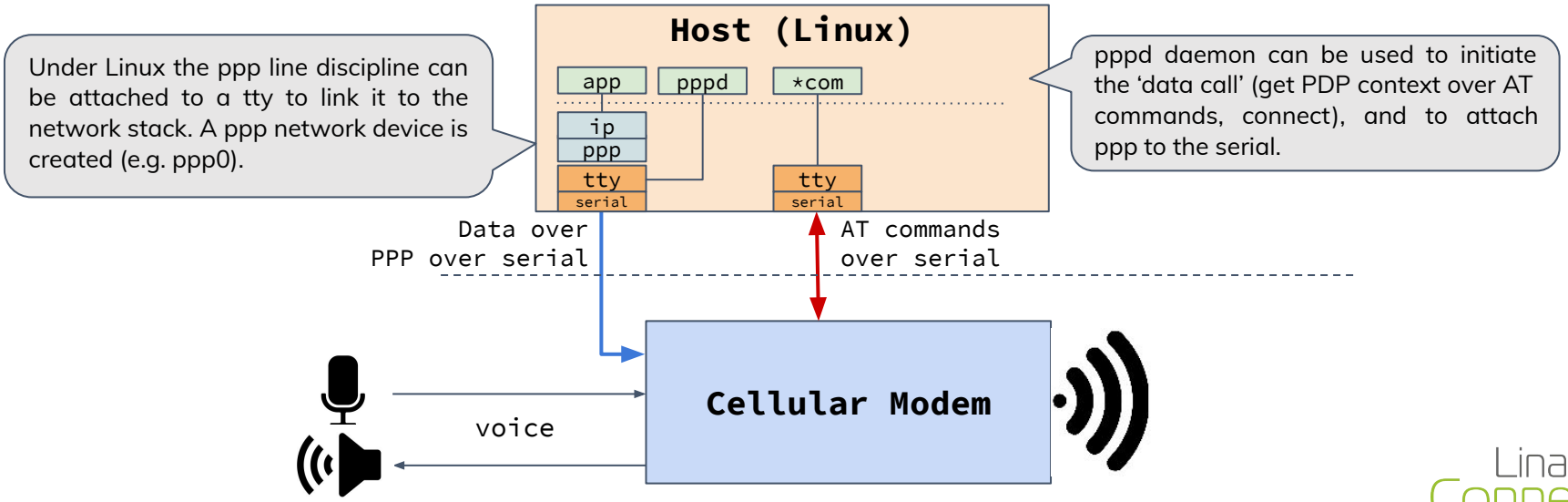
Controlling the modem - AT commands

Host was only in charge of **controlling** the modem, for dialling a number, quieting the speaker, hanging up, dialling a number, quieting the speaker, hanging up, etc. Hayes developed and published a command set to control a modem over a **serial line,** such as UART. This ASCII based command set, known as "**AT command set**" became popular and a defacto standard among modem manufacturers.



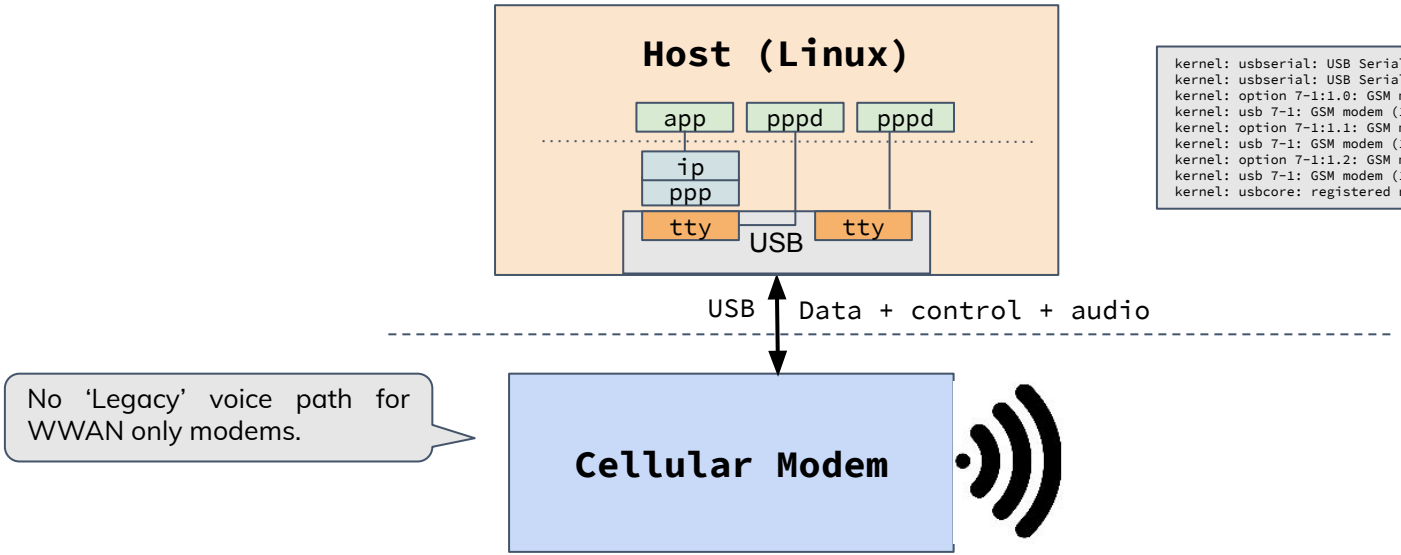
Transferring data path - PPP

When **data** support came to light with 2.5G (GPRS) offering WWAN capability to cellular modems, it was mostly as an **addon feature**. Once A PDP context is established (to start data packet session, provide IMSI, get an address...), PPP (Point-to-Point Protocol) can be used as a data link protocol for transporting datagrams (IP) over the serial link (or dedicated secondary serial link).



USB WWAN devices

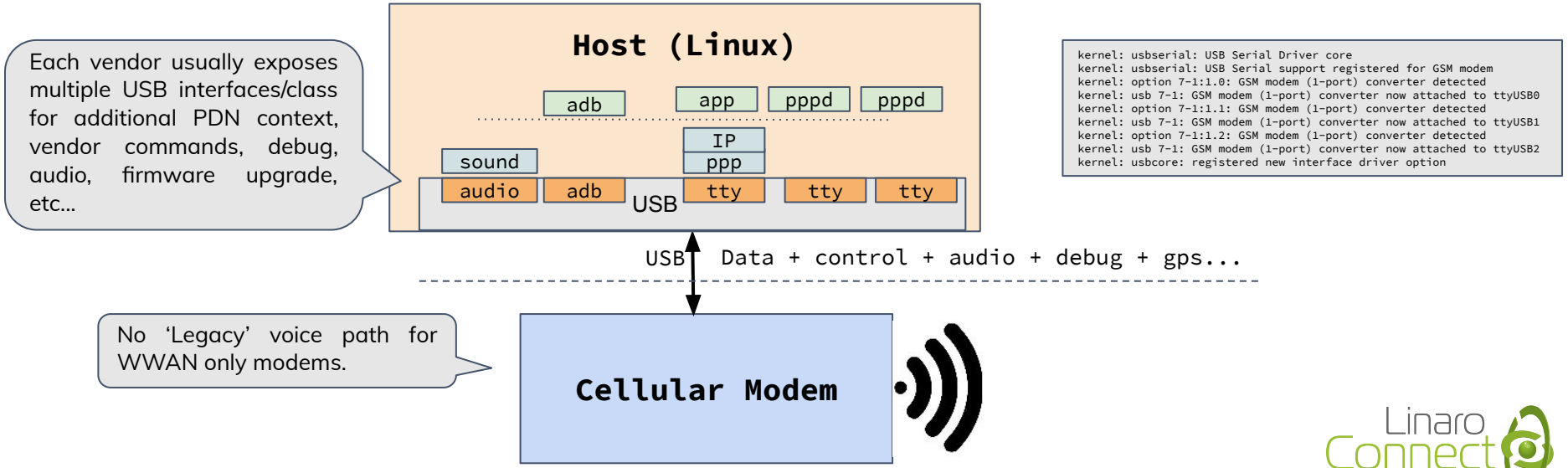
With the arrival of 3G and WWAN democratization, mobile network operators started to offer plans and wireless modems that enable computers to connect to and access the internet , typically in the form of a small USB based device (USB stick, PCMCIA card...). Replicating the legacy serial architecture (AT commands, ppp) over USB was the obvious and simplest choice.




```
kernel: usbserial: USB Serial Driver core
kernel: usbserial: USB Serial support registered for GSM modem
kernel: option 7-1:1.0: GSM modem (1-port) converter detected
kernel: usb 7-1: GSM modem (1-port) converter now attached to ttyUSB0
kernel: option 7-1:1.1: GSM modem (1-port) converter detected
kernel: usb 7-1: GSM modem (1-port) converter now attached to ttyUSB1
kernel: option 7-1:1.2: GSM modem (1-port) converter detected
kernel: usb 7-1: GSM modem (1-port) converter now attached to ttyUSB2
kernel: usbcore: registered new interface driver option
```

USB WWAN devices

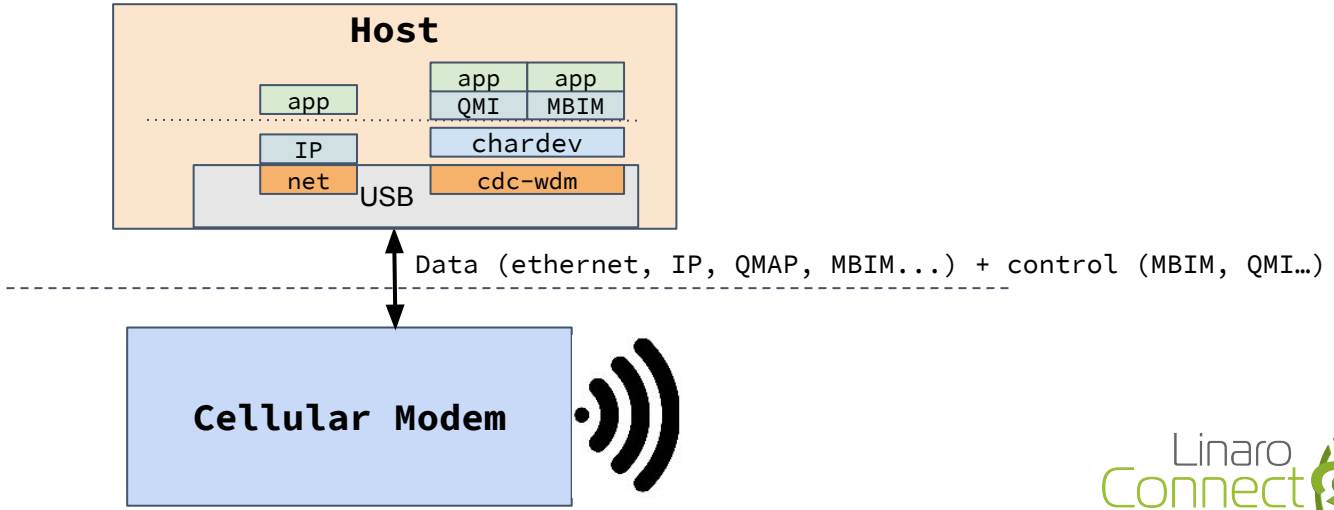
With the arrival of 3G and WWAN democratization, mobile network operators started to offer plans and wireless modems that enable computers to connect to and access the internet , typically in the form of a small USB based device (USB stick, PCMCIA card...). Replicating the legacy serial architecture (AT commands, ppp) over USB was the obvious and simplest choice.




```
kernel: usbserial: USB Serial Driver core
kernel: usbserial: USB Serial support registered for GSM modem
kernel: option 7-1:1.0: GSM modem (1-port) converter detected
kernel: usb 7-1: GSM modem (1-port) converter now attached to ttyUSB0
kernel: option 7-1:1.1: GSM modem (1-port) converter detected
kernel: usb 7-1: GSM modem (1-port) converter now attached to ttyUSB1
kernel: option 7-1:1.2: GSM modem (1-port) converter detected
kernel: usb 7-1: GSM modem (1-port) converter now attached to ttyUSB2
kernel: usbcore: registered new interface driver option
```

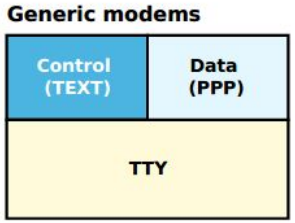
Modern data/control protocols

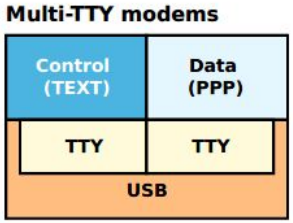
But serial/tty layer is not really optimized for transferring network packets, so vendors started to directly expose **wwan network interfaces** instead (based on CDC-ECM, CDC_NCM or CDC-MBIM USB classes). In the same way TEXT based AT commands protocol is quite limited and started to be replaced (or extended) with **optimized 'binary' control protocols** (offering commands, events and services discovery) such as MBIM (USB Mobile Broadband Interface Model), QMI (Qualcomm), etc.

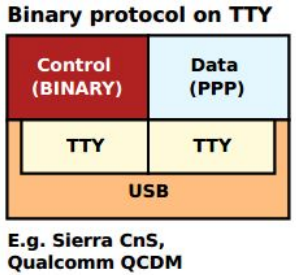


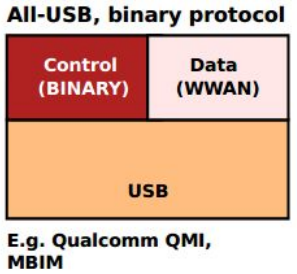
Modem interfaces fragmentation

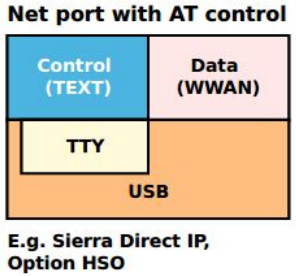
Result of this evolution is a fragmentation of modem interfaces and protocols...

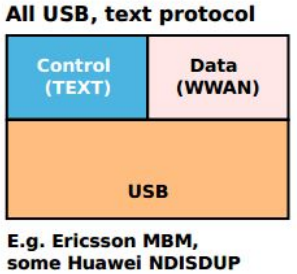












And most modems mix several of these interfaces, ppp+wwan, AT+QMI, etc...

source: https://aleksander.es

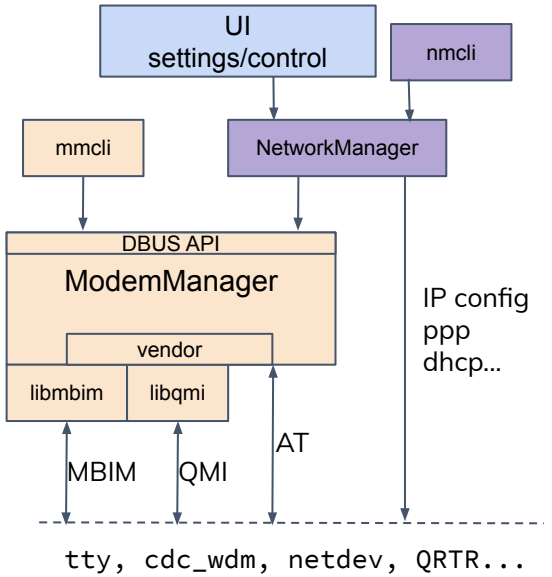
ModemManager to the rescue

ModemManager daemon provides a unified high level API (DBUS) for communicating with mobile broadband modems, regardless of the protocol used to communicate with the actual device (Generic AT, vendor-specific AT, QCDM, QMI, MBIM...).

It is possible to use `**mmcli**` command line interface to control it from the terminal, but it is usually aimed to be controlled by a system **Network daemon** (e.g. NetworkManager, connman...).

ModemManager constructs and exposes **'modem' instance(s)** based on the various devices that contribute to the WWAN feature (based on udev, sysfs device hierarchy...).

ModemManager relies on various vendor plugins for handling **vendor specific device setup/commands** (Huawei, Sierra, Telit, Quectel, Nokia...).



MM

```
$ mmcli -m 1
  ----------------------------
  General  |          path: /org/freedesktop/ModemManager1/Modem/1
       |       device id: c83446b3490da87d10b6f9d9e5a12932cf0ffff
  ----------------------------
  Hardware |   manufacturer: Telit
       |        model: FN980m
       |       firmware revision: XXXXXXX
       |       carrier config: default
       |       h/w revision: 0.00
       |       supported: cdma-evdo, gsm-umts, lte, 5gnr
       |        current: gsm-umts, lte, 5gnr
       |       equipment id: 359661109997912
  ----------------------------
  System   |          device: /sys/devices/pci0000:00/0000:00:14.0/usb4/4-2
       |              drivers: option, qmi_wwan
       |              plugin: telit
       |       primary port: cdc-wdm1
       |              ports: cdc-wdm1 (qmi), ttyUSB0 (ignored), ttyUSB1 (ignored),
       |                     ttyUSB2 (at), ttyUSB3 (at), ttyUSB4 (ignored), wwp0s20u2i2 (net)
  ----------------------------
  Status   |          state: failed
       |       failed reason: sim-missing
       |       power state: on
       |       signal quality: 0% (cached)
  ----------------------------
  Modes |       supported: allowed: 2g; preferred: none
[...]
       |              current: allowed: any; preferred: none
  ----------------------------
  Bands |       supported: utran-1, utran-3, utran-4, utran-6, utran-5, utran-8,
       |                     eutran-66, eutran-71, utran-19
  ----------------------------
[...]
```

5G modem integration story

Qualcomm PCI modems

Qualcomm modems are at the core of many vendor modem modules (Telit, Quectel, Sierra...), especially for new 5G products.

PCI modem variants (Mini PCI, M2 module...) are more and more popular as integrated WWAN solution for consumer laptop, gateways, industrial machines...

PCI usually offers better throughput, power efficiency and lower latency than USB (for same generation), matching new 5G requirements.

Problem: There is no Linux mainline support for Qualcomm-based PCI modems.

Source: https://www.telit.com/mobile-broadband

source: https://www.quectel.com

Modem Host Interface(MHI)

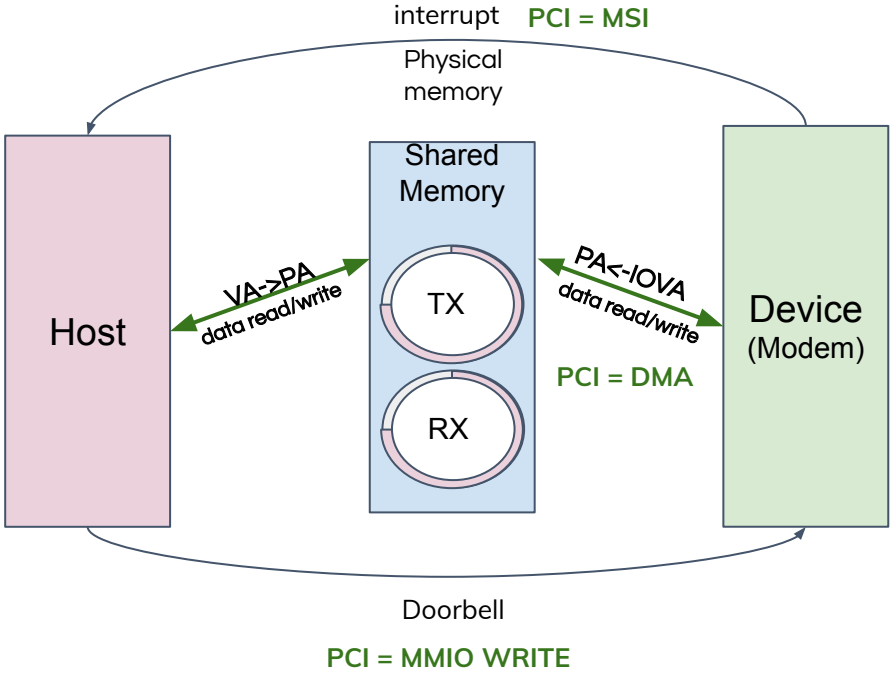
MHI is a protocol initially developed by Qualcomm for interfacing host with PCI modems.

Data transported over **shared memory** using **descriptor rings** and **interrupts/doorbells** for synchronization (similar to virtio).

Concept of **channels** (logical link), states, boot procedure, transfers, etc...

No strictly coupled to PCI, but offer an infrastructure on top of PCI bus low level operations (io-read / io-write / dma / interrupt).

Main interface for all recent Qualcomm modems (SDX20, SDX24, SDX55, etc).



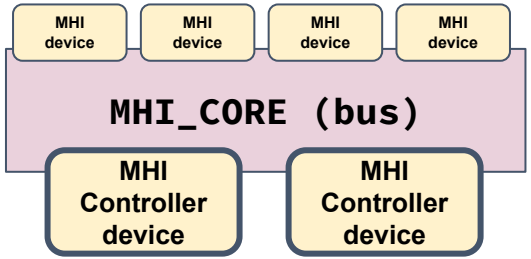
Linux & MHI

First tentative for upstreaming MHI stack in 2018, but discontinued after few iterations.

In 2020, work has been revived by Linaro, Manivannan Sadhasivam addressed the concerns raised by the upstream maintainers for the initial submission and MHI core finally got merged:
https://www.linaro.org/blog/mhi-bus-support-gets-added-to-the-linux-kernel/

MHI has been pushed as a new **virtual bus**, implementing the MHI specification (device initialization and management, transfers...).

A **MHI controller** implements the physical bus operations (read, write...) and register to MHI bus (phy agnostic). the MHI bus exposes MHI channels as **MHI devices**.



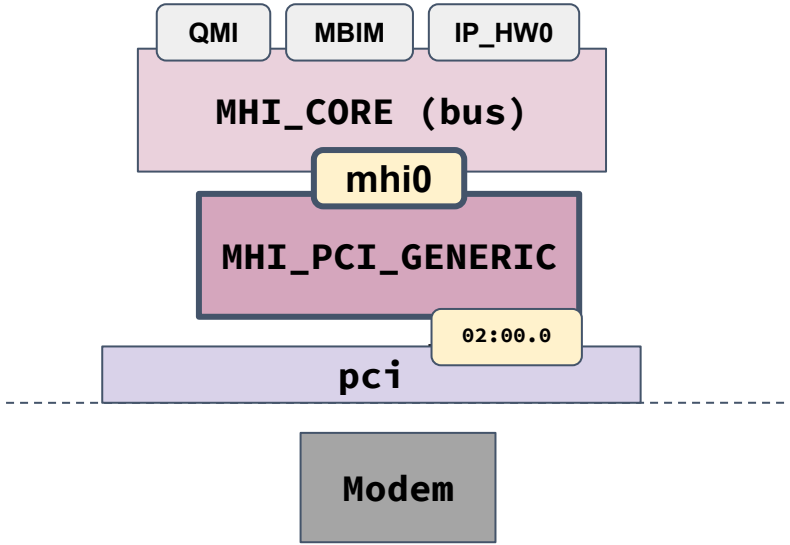
MHI Modem support - PCI controller

mhi_pci_generic is a standard **PCI driver** bound to qualcomm PCI modems (based on hardware IDs).

Basically registering as a **MHI controller**, abstracting physical bus access operations (read/write) for MHI core, and that's it!

MHI core initializes the MHI controller, enumerates the channels, and exposes them as new MHI devices.

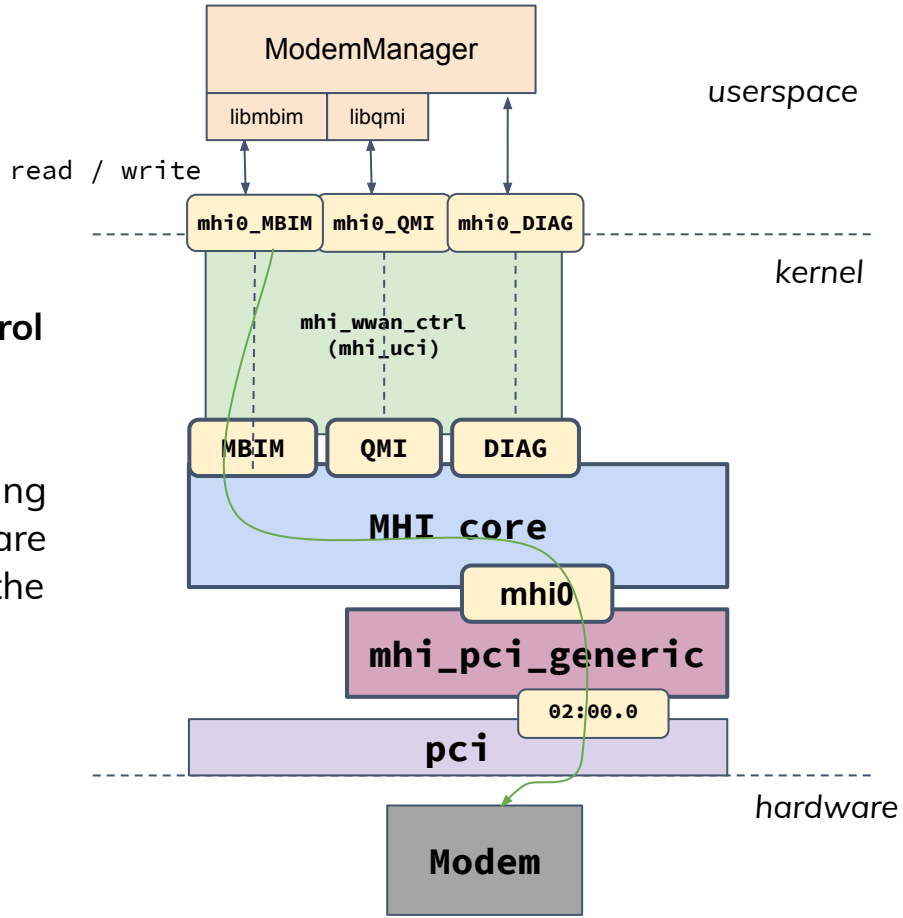
QCOM MHI modems expose multiple channels, including control channels (QMI, MBIM) and data channel(s) (IP_HW0).



MHI Modem support - Control channels

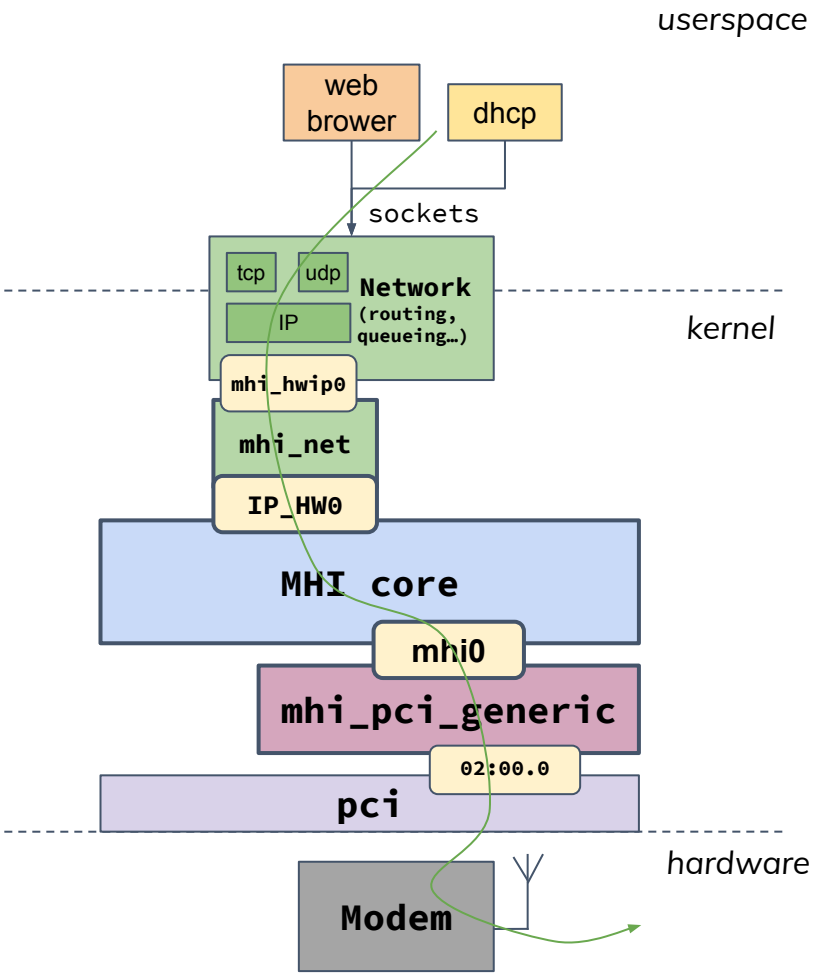
Next step was to create driver for the **control channels** (mhi_wwan_ctrl/mhi_uci).

In order to facilitate compatibility with existing userspace tools and stack, control channels are exposed as standard **character devices**, in the same way as for USB modem variants.



MHI Modem support - Control channels

On the other side, **data channel** is registered as a standard **network device** via mhi_net driver.



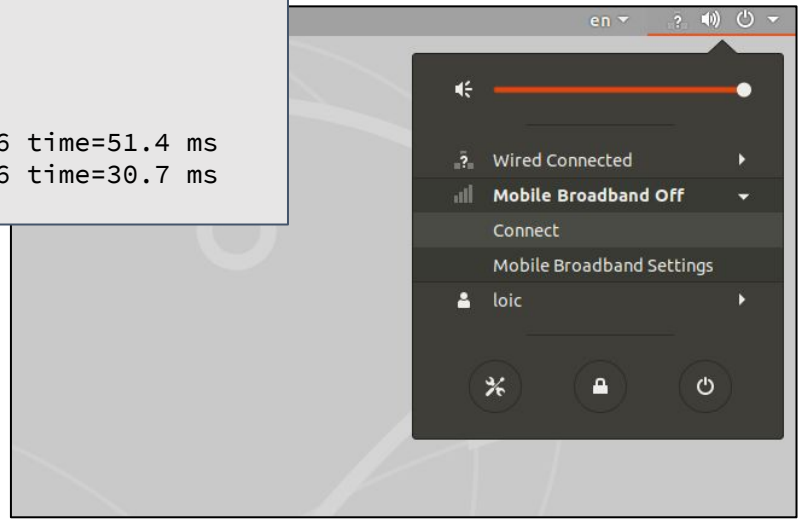
Integration with ModemManager

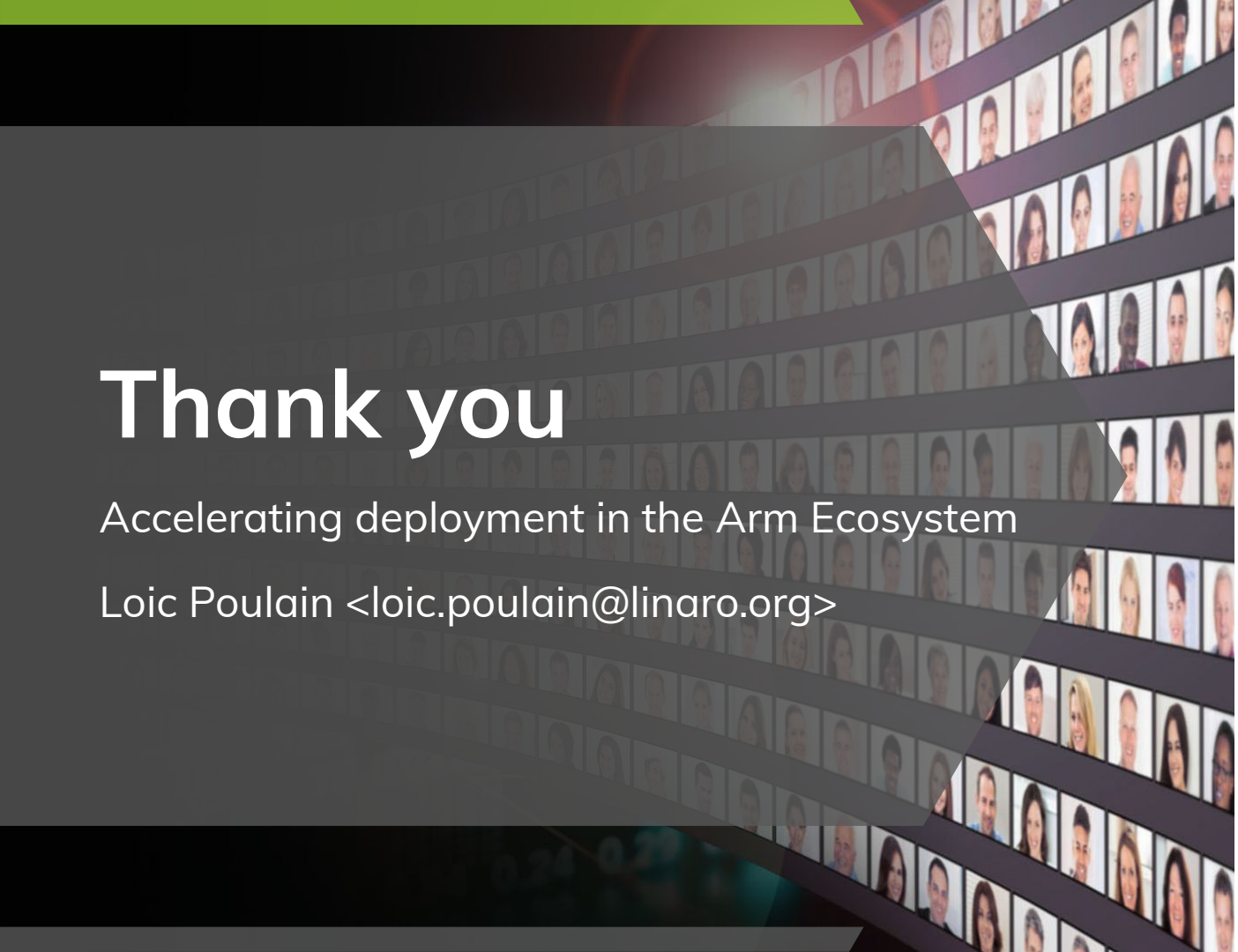
```
$ mmcli -m 0
  ------------------------------
  General   |           path: /org/freedesktop/ModemManager1/Modem/0
            |     device id: 2ead7f1403aeda3de2a775b50c15976ccf3ba521
  ------------------------------
  Hardware  |     manufacturer: Telit
            |            model: FN980m
            |  firmware revision: XXXX
            |     carrier config: default
            |     h/w revision: 0.00
            |           supported: cdma-evdo, gsm-umts, lte, 5gnr
            |           current: gsm-umts, lte, 5gnr
            |     equipment id: 359661109997912
  ------------------------------
  System    |           device: /sys/devices/pci0000:00/0000:00:01.2/0000:02:00.0
            |           drivers: mhi_net, mhi_uci
            |           plugin: generic
            |     primary port: mhi0_QMI
            |           ports: mhi0_QMI (qmi), mhi_hwip0 (net)
  ------------------------------
  SIM       |  primary sim path: /org/freedesktop/ModemManager1/SIM/0
            |     sim slot paths: slot 1: /org/freedesktop/ModemManager1/SIM/0 (active)
```

And voilà!

```
$ sudo nmcli c add type gsm ifname mhi0_QMI con-name MY_CELLULAR_CON \
                  gsm.apn free gsm.pin 1234 connection.autoconnect yes
Connection successfully activated

$ ping google.fr
PING google.fr (216.58.204.99) 56(84) bytes of data.
64 bytes from par10s28-in-f3.1e100.net: icmp_seq=1 ttl=116 time=51.4 ms
64 bytes from par10s28-in-f3.1e100.net: icmp_seq=2 ttl=116 time=30.7 ms
```





Thank you

Accelerating deployment in the Arm Ecosystem

Loic Poulain <loic.poulain@linaro.org>

Linaro Connect
VIRTUAL • SPRING 2021