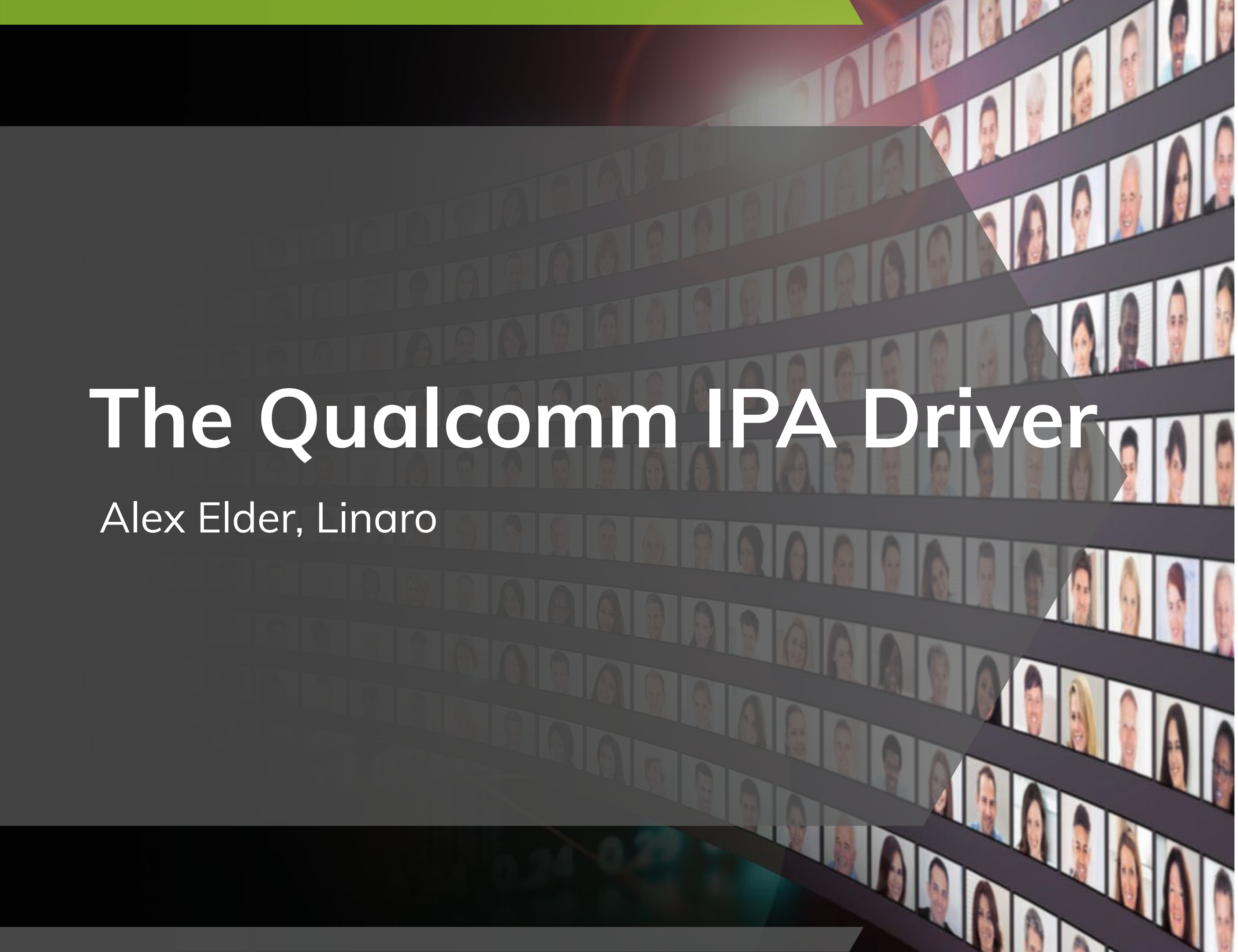


The Qualcomm IPA Driver

Alex Elder, Linaro

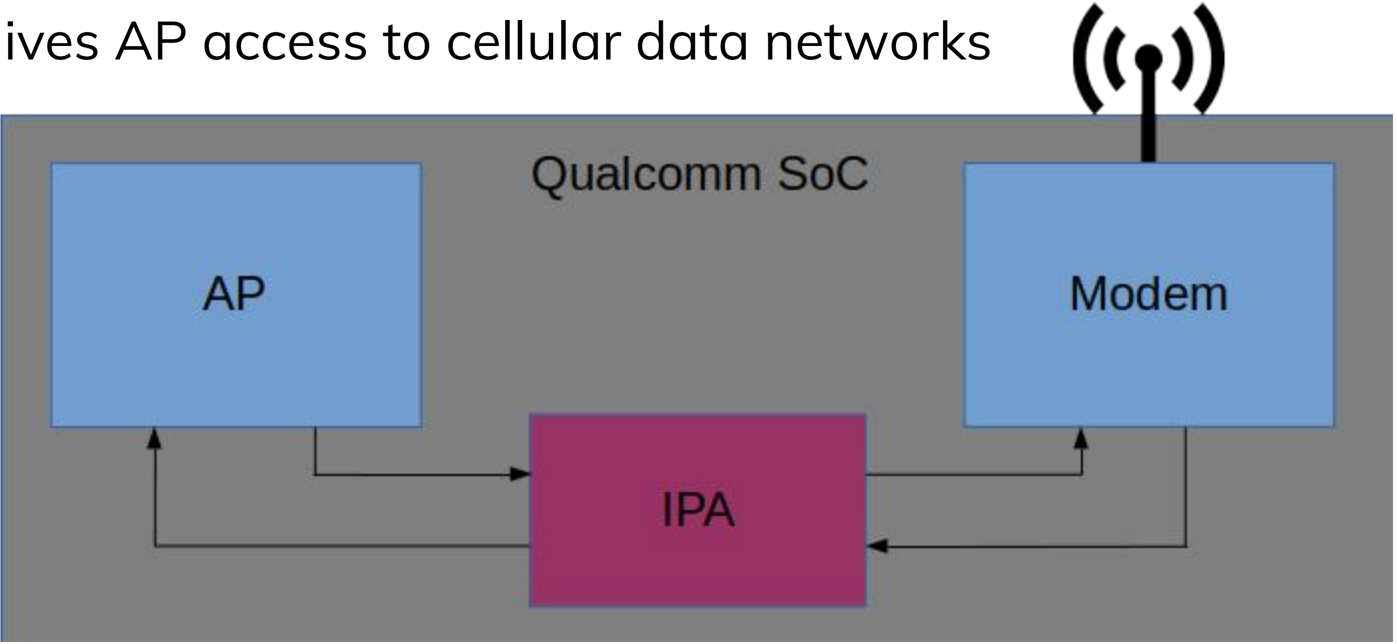
Linaro Connect
VIRTUAL • SPRING 2021

Introduction

- What is IPA?
- Problem statement and approach
- Work to be done
- Challenges along the way
- Current status
- Some insights and lessons learned
- Summary

IPA Overview

- SoC resident network switch
- Gives AP access to cellular data networks



Linaro Connect VIRTUAL • SPRING 2021

IPA Capabilities

- Presented by its driver as a network device
- Performs checksum offload, packet aggregation
  - Reduces processing and interrupt load on the main CPU
- Also implements integrated IPA filtering, routing, and NAT
  - These features are not supported by the upstream driver (yet!)
- Capable of operation independent while AP is asleep
  - Tethered operation (WiFi hotspot)
  - Requires much less power than operating AP
  - This mode is not supported upstream either

Problem Statement

- Qualcomm has downstream code to support their hardware
  - This is open source code, but “out of tree” (not upstream)
- The “downstream” code is hosted on the Code Aurora Forum
- Qualcomm hardware is supported for some environments
  - Windows, Android, other Linux-based environments
- Desire to expand to support Chrome OS also
  - But Chrome OS does not accept out-of-tree code
  - The code must be upstream (or clearly on its way there)
- So the objective was:

  Upstream the IPA driver

Early days

- Qualcomm asked Arnd Bergmann for his assessment of the driver
    - He said the IPA driver was too complex for the upstream kernel
- Qualcomm created a reduced functionality driver
    - No filtering, routing, NAT
    - Only support “embedded” network connection to the AP
    - 100,000+ lines of code reduced by over half
- The result was “IPA lite”
    - Provided basic functionality required for Chrome OS
    - Known to work on internal hardware
    - It was a subset of their downstream driver

Expectations

- Qualcomm wanted IPA lite to be their upstream driver
  - Wanted to avoid having to maintain two separate code bases
  - Expected to adjust their downstream code as needed
  - Then enhance IPA lite to add back functionality once upstream
- Expectations were very optimistic
  - IPA lite had considerably less functionality than the “full” driver
  - Hope was to get the code accepted without major changes
  - Expected time frame was a matter of months

Reality

- IPA lite was not acceptable for upstream
  - Despite incorporating some very good design
- To get upstream:
  - Someone needs to review the code
  - Someone needs to **accept** the code
- Reducing the functionality **was** very important
  - It did remove some high-level complexity
  - It also reduced the sheer amount of code
    - Nobody wants to review an enormous driver
- But much more would have to change
  - Lots of superficial and somewhat mechanical fixes
  - Other design issues needed to be resolved

Superficial problems

- Coding style
  - Large functions, inconsistent indentation
  - Long symbol names, CamelCase
- Duplicated code, in need of refactoring
- Dead code
  - Or at least code that is unused for the target platform
- Excessive use if #ifdef

Design problems

- Using CPU parallelism to serve a single hardware queue
  - Rather than having multiple CPU-pinned hardware queues
- NAPI
  - All new network devices should use NAPI for interrupts
- Overuse of work queues
  - In some cases, threaded interrupts are enough
- Abuse of Device Tree
  - Only platform description should be specified there
- IOCTL interface
  - No longer acceptable
- Hardware abstraction layer
  - Upstream reviewers tend to be skeptical

The Plan

- Upstream development plan:
  - Start with IPA lite driver
  - Modify the driver iteratively, evolving it toward “upstream ready”
  - Post for upstream review as soon as “reasonable”
- All parties agreed to this strategy
  - But “upstream ready” and “reasonable” were perhaps interpreted in more than one way

Initial challenges

- Each phase of development brought its own challenges
- The code was initially only available inside the Qualcomm firewall
    - Windows system was required for access
- No hardware was available for testing
    - This meant greater care required for code changes
- Schedule expectations were unrealistic
    - Conflict arose because of:
        - Qualcomm’s desire to complete on schedule
        - My desire to post high quality code for upstream review
    - Resetting expectations early might have helped

Early upstream challenges

- Code was posted for RFC review in November 2018
- Got some good feedback, representing lots of work to do
    - Don’t use global variables
    - Don’t use “register HAL” if possible
    - Use BQL (byte queue limits); look at CoDel and RFC 8289
    - Use NAPI
    - Avoid additional queueing
    - Avoid excessive locking
    - Don’t implement wrappers around well-defined kernel code
    - Avoid indirect function calls (Spectre)
    - Don’t use IOCTL and don’t use BUG()

Later upstream challenges

- First “real” upstream post of the code was in May 2019
- WWAN framework
  - A proposal to unify the representation of a wireless WAN device in the kernel
  - Request: define the WWAN framework first
- RMNet over IPA
  - Why is the “rmnet” driver needed (layered above IPA)
  - Buffer bloat concerns

Upstream challenges

- The accepted set of patches were posted in March, 2020
  - No WWAN framework
  - RMNet driver remains layered on top of IPA
- This code is considerably different from IPA lite
  - And therefore quite different from downstream IPA
  - Two code bases after all (but I maintain one of them)
- Downstream code has continued to evolve
  - No longer familiar
- Bug fixes aren't likely to be shared between code bases

Some insights

- Code quality requirements
  - Code is not accepted upstream on a schedule
    - It must be “upstream quality” and should be tested
  - Downstream **must** meet schedule
    - It should be “good enough” and must pass all tests
- Software lifetime
  - The upstream kernel moves forward, continuously
  - Downstream freezes a release for a platform
- Deprecation
  - Old hardware must continue to work upstream
  - Downstream can ignore support for old hardware

Code partitioning

- The entire upstream kernel source tree is a unit
    - Any developer can touch any part of the kernel
    - The primary “stable API” is the one presented to user space
    - All users of a symbol can be known
- An organization like Qualcomm divides responsibility
    - Different teams “own” different parts of the kernel
    - Important to preserve stable ABIs **within** the kernel
    - Can’t assume anything about users of a symbol
        - Callers of a function might supply garbage

More Insights

- Assumptions about hardware
  - Upstream code generally assumes working hardware
  - Downstream code is used for hardware bringup
- There are debugging differences too
  - Voluminous debug output can help diagnose problems quickly
  - But it’s overkill for normal upstream needs
- Development processes
  - Linux development has a fairly well-defined model
  - Linux is not the only environment for places like Qualcomm

Current state

- Platforms currently supported upstream
  - SDM845, SC7180 (Snapdragon 7c)
  - Support for a third (and more) will be coming soon
- IPA access to a cellular modem is usable on Arm64 based laptops
  - Still some work to do with user space integration
- Work in the coming year
  - Performance tuning
  - Adding support for advanced features (filtering, routing, NAT)
  - Work toward tethering support

Summary

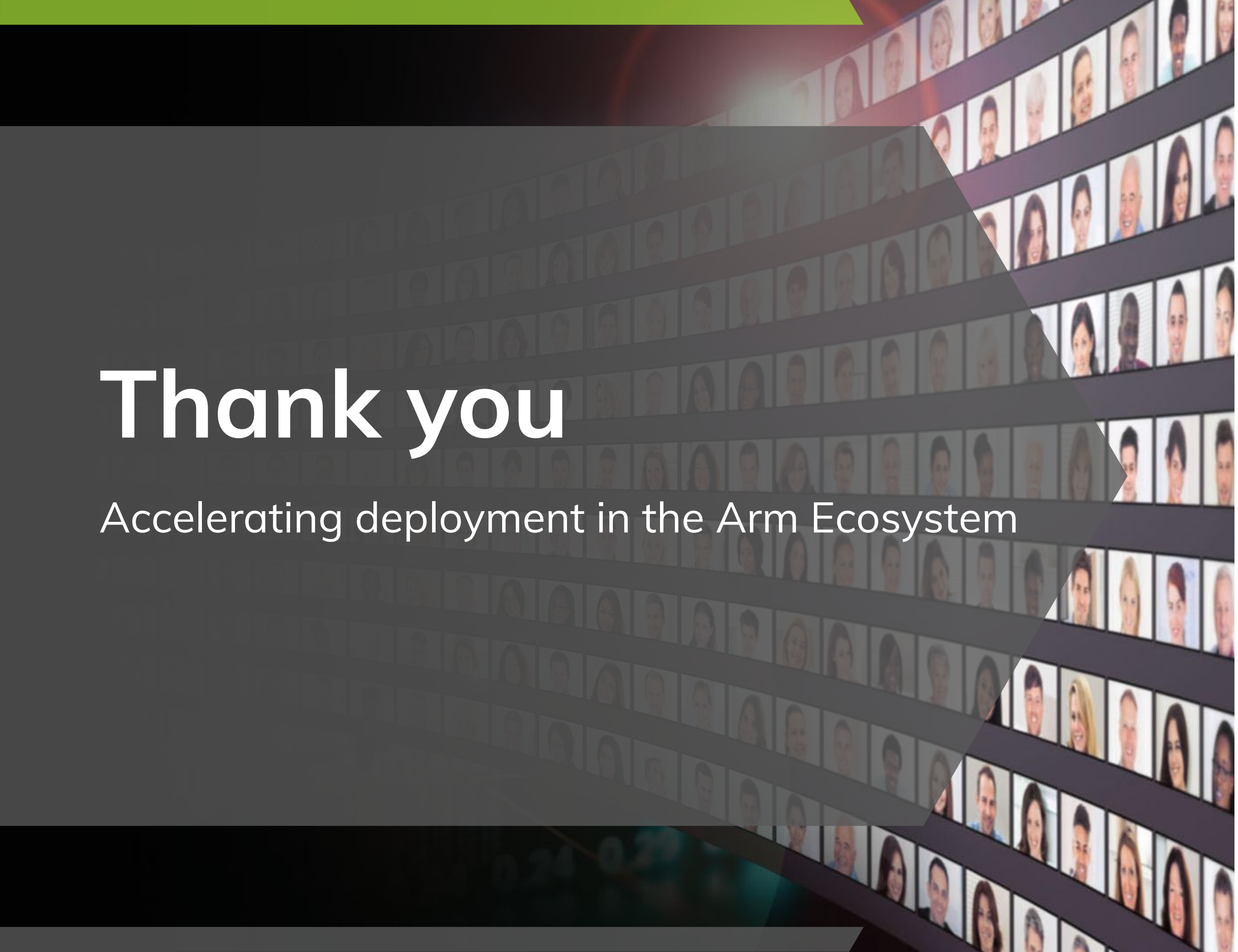
- The Qualcomm IPA driver is upstream!
  - But getting there wasn’t easy
- To get code upstream, it needs to be reviewed and accepted
  - It needs to at least “look like” upstream code
  - Someone must be willing to review it
- Upstream code standards are very demanding
  - Even superficial problems can preclude acceptance
  - But other design issues **will be found** in review
- Downstream and upstream code have different requirements
  - It’s no surprise that adapting downstream code takes work

Acknowledgements

- Qualcomm IPA team
  - Praveen, Ashok, Chaitanya, Susheel, Vaibhav, Dan, Gopi, Abhishek, others
- Arnd Bergmann
  - For actually reviewing the driver, and providing valuable feedback and insight
- The Linux network maintainers
  - David Miller, Jakub Kicinski, and others do an amazing amount of review
- The Chrome OS kernel team at Google
  - Evan, Sujit, Matthias, Eric, Stephen, Rob, Doug, Ryan, Grace, others
- Linaro
  - The company and the individuals in it make it easy to get things done

References

- Downstream IPA code at the Code Aurora Forum
  https://source.codeaurora.org/quic/la/platform/vendor/opensource/dataipa/
- ChromeOS upstream first policy
  https://www.chromium.org/chromium-os/chromiumos-design-docs/upstream-first
- RFC upstream post, November 2018
  https://lore.kernel.org/netdev/20181107003250.5832-1-elder@linaro.org/
- Initial upstream posts, May 2019
  https://lore.kernel.org/lkml/20190512012508.10608-1-elder@linaro.org/
  https://lore.kernel.org/netdev/20190531035348.7194-1-elder@linaro.org/
- Accepted version, March 2020
  https://lore.kernel.org/netdev/20200306042831.17827-1-elder@linaro.org/



Thank you

Accelerating deployment in the Arm Ecosystem

Linaro Connect
VIRTUAL • SPRING 2021