The *Serverkernel* Operating System

Jon Larrea and Antonio Barbalance

University of Edinburgh

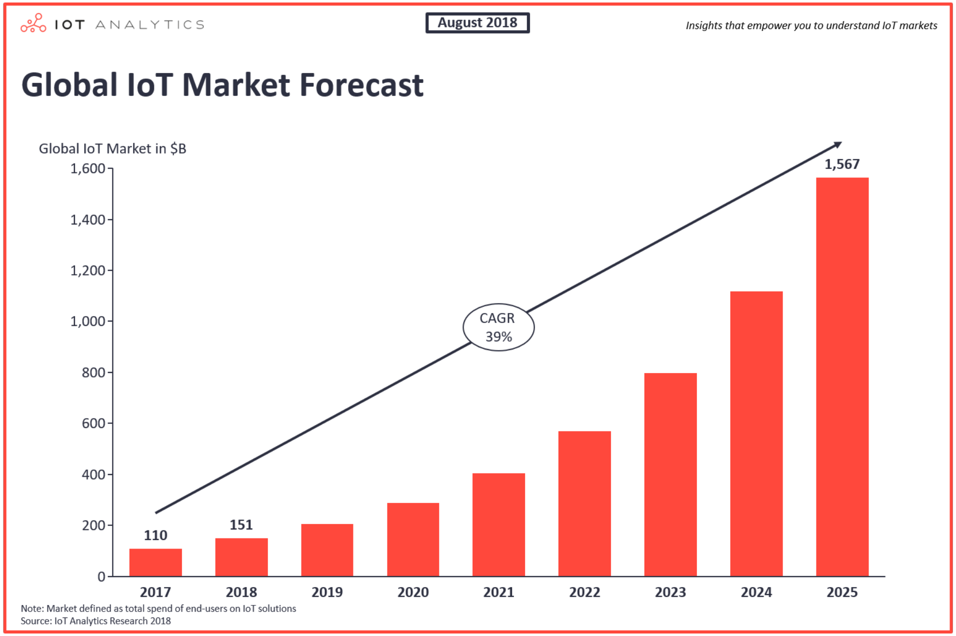
April 23, 2020

Table of content

- Introduction
  - Current IoT environment
- Architecture
  - Design
  - Background
  - Operating Principles
- Implementation
  - jonOS
  - Modules
  - Client Toolchain
- Initial Results
  - CPU
  - Network
- Open Research Problems

IoT Growth

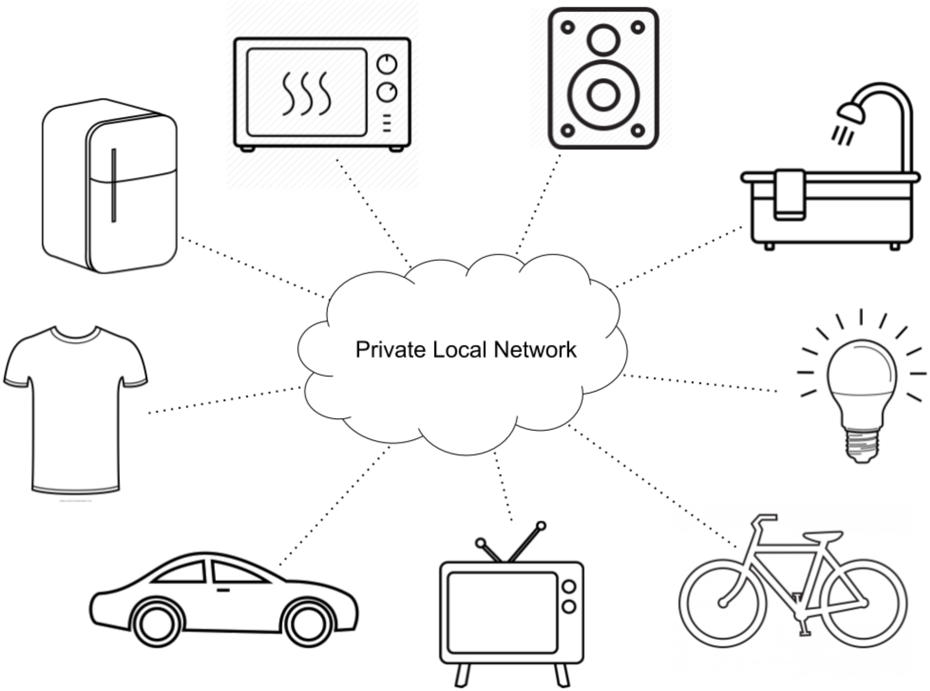
- Exponential growth in the number of interconnected devices
- Improvements in performance, miniaturization, energy consumption
- Per-device price dropped down



Note: Market defined as total spend of end-users on IoT solutions
Source: IoT Analytics Research 2018

Current IoT environment

- IoT devices are part of our life, Integrated into everyday objects
- Set of inter-networked processors that remain most of the time in idle status



The *Serverkernel*

A new OS kernel design

- Based on the principle of **extreme minimality** for
  - High performance
  - Energy conservation
- Targets **IoT and generic embedded devices**
  - With (any) network connectivity
  - Mostly idle
- Enables **secure compute offload** on those
  - Via network
  - When idle

The *Serverkernel* Architecture

A **single-address space mono-task OS** that only supports a limited set of functionalities

- Borrows ideas from several previous works
  - ***Exokernel:*** Serverkernel is based on a *libOS*
    - minimal access time to hardware resources
  - ***Unikernel:*** Application runs at kernel-level
    - avoid syscall overhead
  - ***RTOS:*** Code is written for bounded latency
    - predictable single user execution
  - ***Function as a Service (FaaS):*** Offload pieces of apps
    - execute the code sent by 3rd party applications

Operating Principles

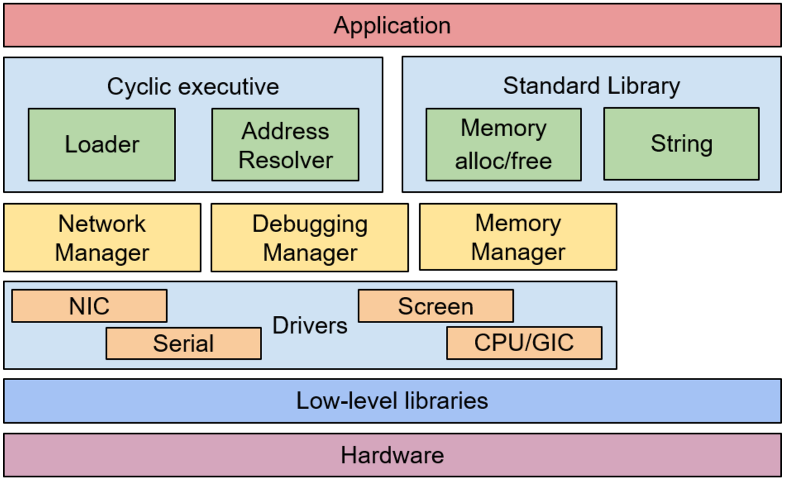
To run a piece of an applications on a remote device that runs a *Serverkernel* the application has to

1. Identify a reachable *Serverkernel* on the network
2. Authenticate the *Serverkernel*
3. Compile the code in the advertised *Serverkernel* format
4. Establish a secure connection with it
5. Send the application code
6. Wait for the result

jonOS

A **modular** and **open-source** implementation (C and asm) of the *Serverkernel*

- Integrates essential *Serverkernel* functionalities
  - Runs on bare-metal ARM boards BCM2835-based
- Includes a toolchain to create *jonOS* executable binaries
  - Based on Python and GNU GCC



jonOS Modules

A **module** is an OS service, device driver, library, etc.

- **OS Services**
    - *Network Manager*
    - *Debugging Manager*
    - *Memory Manager*
    - *Cyclic Executive*
- **Drivers**
    - *Network*: UDP/IP stack
    - *Serial*: UART
    - *Screen*: HDMI
    - *CPU/GIC*
- **Libraries**
    - *Standard library*

jonOS Toolchain

Based on the *arm-none-eabi* cross compiler

Provides the following guarantees:

- ***Serverkernel* executable format**
  - Binary blobs that follow the spec executable format
  - E.g., position-independent code
- ***Serverkernel* executable loading procedure**
  - System calls have to be resolved at load time
  - *jonOS* resolves system calls addresses in $O(1)$

Initial Evaluation

Performance comparison of *jonOS* vs Linux/Raspbian

Hardware

- Raspberry Pi 1 Model B (device under test)
- Intel x86 workstation (serial line and Eth connection)

Compute Performance

MD5 Hash calculation

- *CPU time*
- *Execution time*

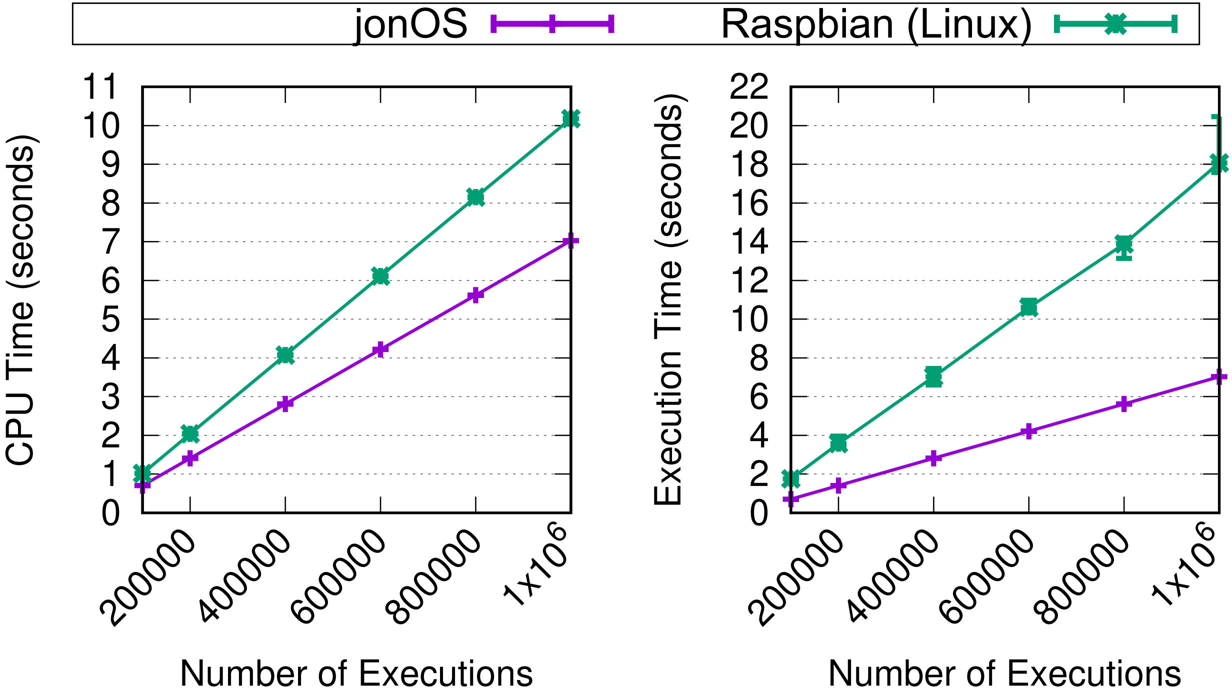
Network Performance

Echo server

- *Response latency*

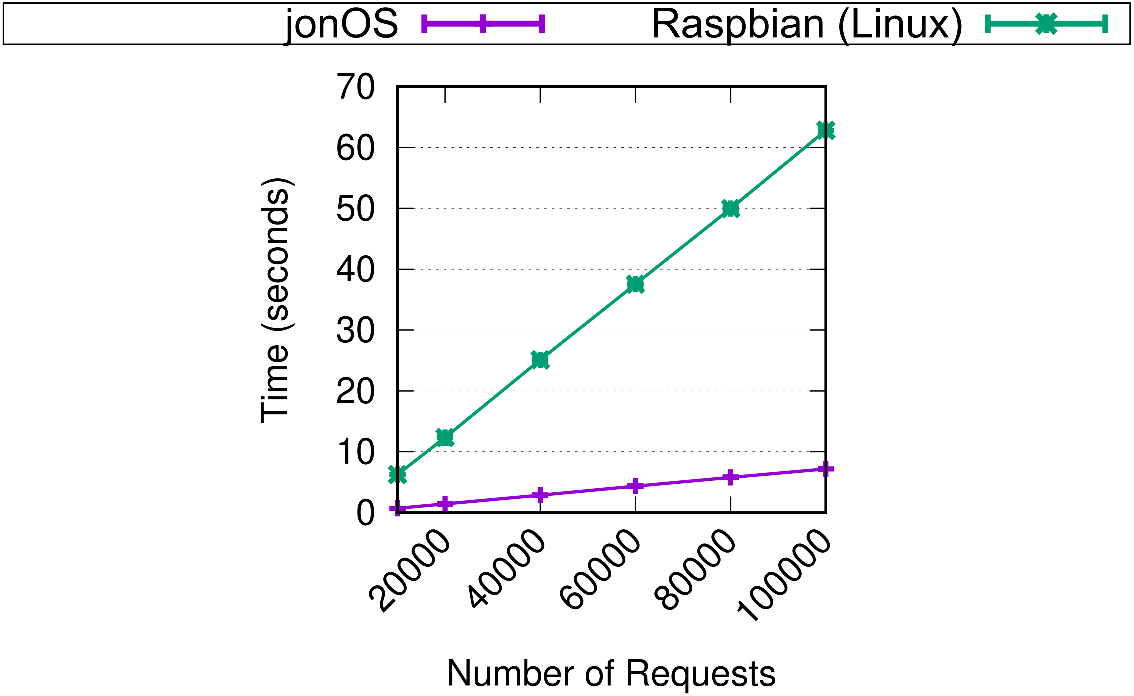
CPU Results

CPU time: *jonOS* shows an improvement of 45.6%
Execution time: *jonOS* the improvement rises up to 62%



Network Results

jonOS performance is almost **9 times better** than Raspbian



What's Next?

Full *Serverkernel* implementation!

- Integration within a real-world IoT device
- Distributed task offloading
- Security
- Full integrated toolchain (in Android, iOS, etc.)

Source code available at: *github.com/j0lama*

s2004865@ed.ac.uk