Actors for the Internet of Things

Pushing CAF to RIOT

Raphael Hiesgen
raphael.hiesgen@haw-hamburg.de
Internet Technologies Group
Hamburg University of Applied Sciences

Agenda

1. General Background
2. Actors for the IoT
3. CAF on RIOT
4. Experimentation
5. Conclusion and Future Work

The Internet of Things (IoT)

- Network of appliances
  - Often constrained embedded devices
  - Act as sensors and actuators
  - Depend on machine-to-machine communication
  - Connected through Internet standards
- Typical communication patterns
  - Data collection: many-to-one
  - Control: one-to-many
- Platform for distributed applications

Problem Statement

- Highly distributed application design
- Development requires specialized knowledge
  - Communication, synchronization and scalability
  - Usually in low-level languages (such as C)
  - Error-prone and hard to debug
- Deployment is platform-specific
- No established programming model

Relevance of Research

- Ease application development
- Reduce the development overhead
- Professionalization
    - Reusability, Robustness, Portability
- Promote experimentally driven research
    - IoT environments often unpredictable
    - Reproducibility is not a given
    - Provide tools to test and deploy software
- Search for the glue of IoT programming

Agenda

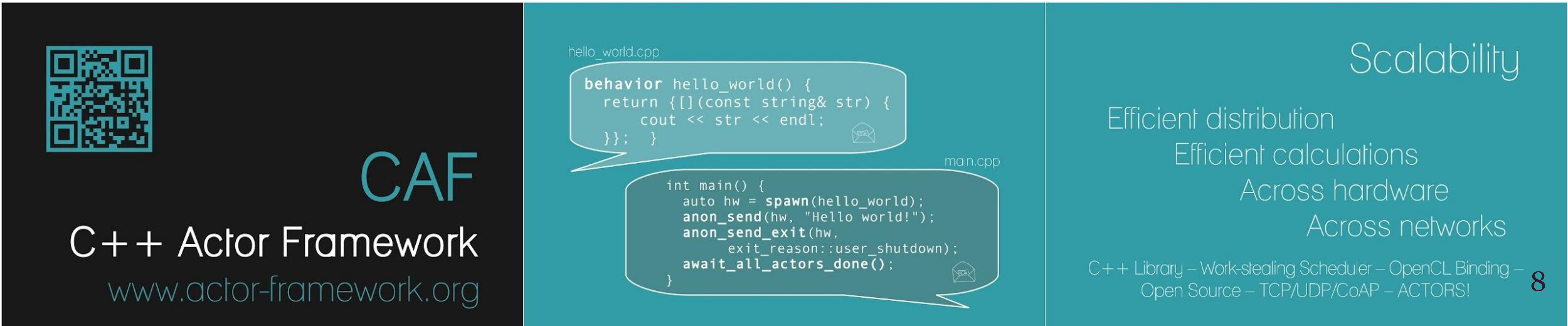
1. General Background
2. Actors for the IoT
3. CAF on RIOT
4. Experimentation
5. Conclusion and Future Work

Approach

- Actors as base entities
  - Run concurrently & in isolation
  - Can spawn new actors
- Distributed runtime environment
  - Network transparent message passing
  - Distributed error-handling
- Network of actors as a design candidate
  - Program distributed applications

The C++ Actor Framework

- Implementation of the actor model
- Available under Revised BSD or Boost license
- Small memory footprint
- Different runtime implementations
  - Memory management & scheduler
- Static type-checking
- Runtime inspection tools



CAF

C++ Actor Framework

www.actor-framework.org

hello_world.cpp

```
behavior hello_world() {
  return {[](const string& str) {
      cout << str << endl;
  }};  }
```

main.cpp

```
int main() {
  auto hw = spawn(hello_world);
  anon_send(hw, "Hello world!");
  anon_send_exit(hw,
       exit_reason::user_shutdown);
  await_all_actors_done();
}
```

Scalability

Efficient distribution

Efficient calculations

Across hardware

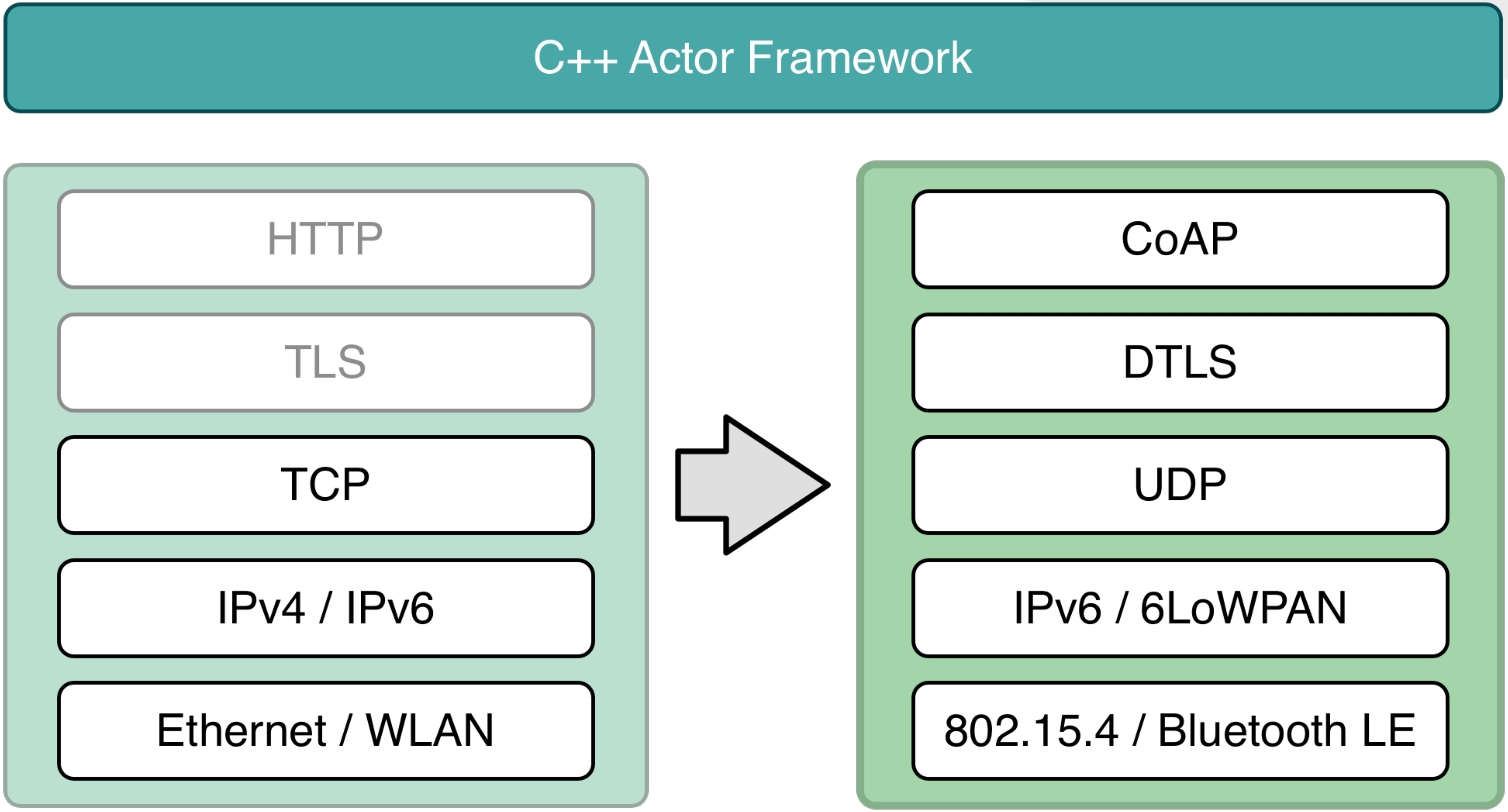
Across networks

C++ Library – Work-stealing Scheduler – OpenCL Binding – Open Source – TCP/UDP/CoAP – ACTORS!

Adaption to the IoT

- Communication protocols
  - Lossy links are common
  - Handle infrastructure failure
- Requires suitable messaging layer
  - Message exchange
  - Synchronization
  - Error propagation and mitigation
- Security
  - Nodes may contain private data
  - Encryption & authentication

Network Stack



C++ Actor Framework

HTTP

TLS

TCP

IPv4 / IPv6

Ethernet / WLAN

CoAP

DTLS

UDP

IPv6 / 6LoWPAN

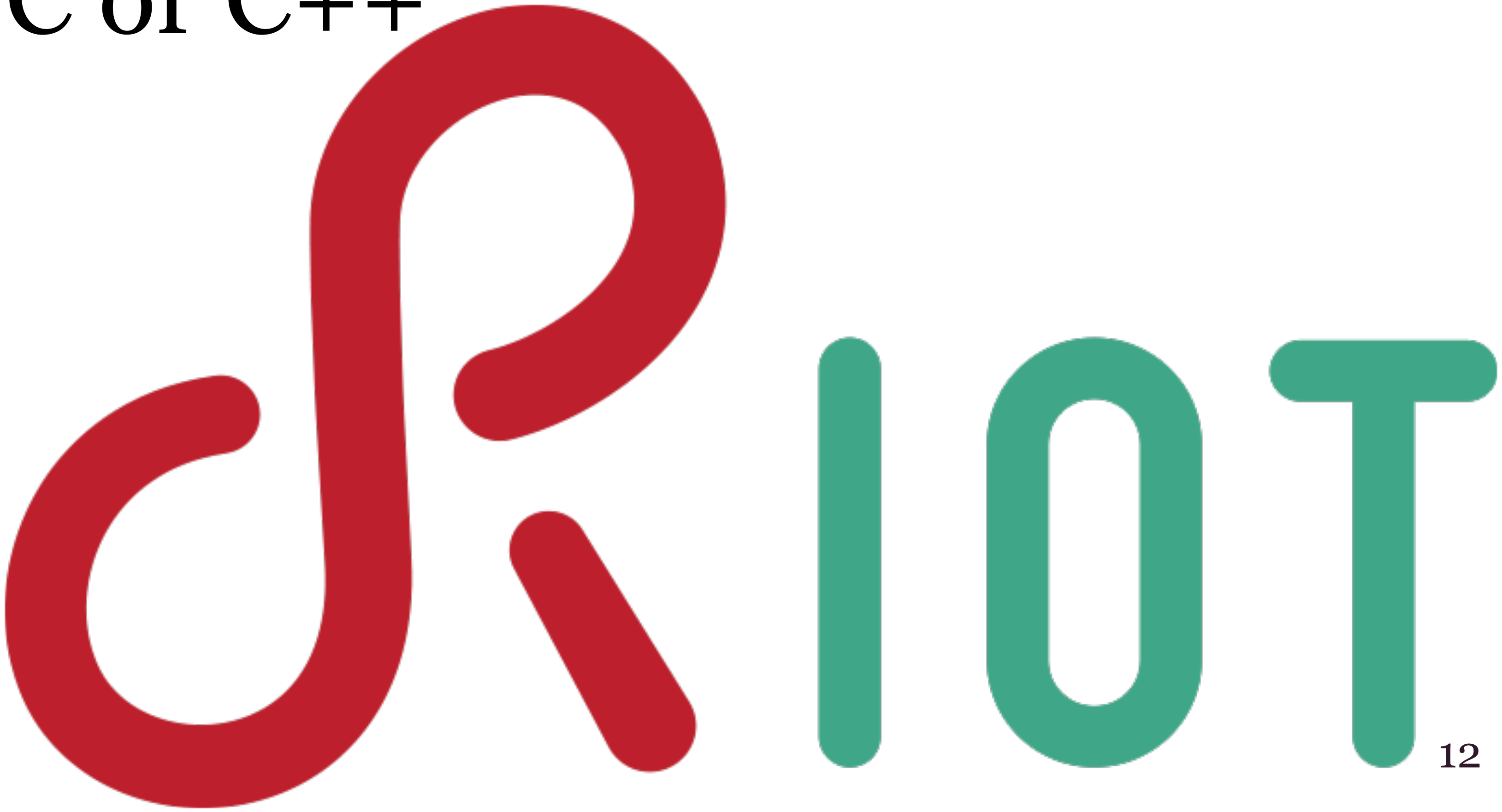
802.15.4 / Bluetooth LE

Transactional Layer

- Transactions
  - Each message exchange is independent
  - Even if it is fragmented
- CoAP
  - Duplicate message detection
  - Reliable message transfer
  - Fragmentation of large messages
- CAF
  - Message header compression
  - Error propagation

Support of Embedded OSs

- The friendly Operating System of the IoT
- POSIX compliance
- Energy efficient
- Real-time capable
- Development in C or C++



Agenda

1. General Background
2. Actors for the IoT
3. CAF on RIOT
4. Experimentation
5. Conclusion and Future Work

Roadmap

- Goal: CAF on RIOT
    - libcaf_core
        - native port (done)
        - stm32f4discovery (WIP)
    - Implement network stack in CAF (open)
    - libcaf_io
        - native port (open)
        - stm32f4discovery (open)
- Takes a surprising amount of time
- Progress can be found on Github
    - Branches are topic/riot and topic/caf

The First Idea

- Let’s use GCC to compile for native
  - Substitute pthread for RIOT’s pthread
  - *“what(): Enable multithreading to use std::thread: Operation not permitted”*
- Dig into GCC source code
  - `if (!__gthread_active_p()) { /* err */ }`
  - Removing the error check helps
- Turned to the libstdc++ mailing list
  - *“Using a custom pthreads implementation is not expected, so it's not surprising if it doesn't work perfectly. (...)”*
- Undesirable workflow anyways

Thread, Mutex and Condition

- Preserve API of the Standard Template Library (STL)
  - Few changes to CAF implementation
  - Familiar to most C++ developers
- Introduce new headers
  - STL or RIOT-based depending on build flag
  - Use caf namespace to prevent ambiguity
  - Omits pthread indirection

```
#ifdef __RIOTBUILD_FLAG
// Our implementation
#else
// Include STL header, provide functions in caf namespace
#endif
```

Getting Threads to Run

- Mostly straight forward (e.g., clang, GCC, ...)
- Implemented thread stack as a member
  - Clang-built executable worked fine
  - GCC-built executable crashed when it entered main
  - Switched GDB to asm mode
  - Stack pointer incremented by an unbelievable amount
- The stack is allocated on the heap
  - A stack on a stack of the same size is a bad idea
  - Detach requires it to be no member
  - Questionable on embedded

How About Locks?

- Removed the destructor of unique_lock
  - Critical for its functionality (release the mutex)
  - My test was an example from the internet
  - Always unlocks the mutex manually (unnecessarily)
- Triggered me to write my own tests
  - Tests for thread, mutex and condition variable
  - Should have done this previously

Compiling CAF for RIOT

- Disabled features
  - Memory Management
  - CAF examples & unit tests
- Changes for the compiler
  - Include modules from RIOT
    - sys, core and cpu
    - Will be linked in a later step
  - Static and 32 Bit
  - Include C files with: `extern “C”`

Static Initialization

- A simple example with CAF on RIOT crashes
    - GDB points to comparison with uninitialized objects
    - These should have been initialized before main
    - Test reveals that static initialization is not working
- GCC offers an array with init functions
    - RIOT startup code never called them
- RIOT mailing list provided a fix for native
    - Only works for native with GCC

GCC Static Initialization

```c
typedef void (*func_ptr)(void);
extern func_ptr __init_array_start[];
extern func_ptr __init_array_end[];
int size = __init_array_end - __init_array_start;
int i, flag = 0;
for (i = 0; i < size; i++) {
    if (__init_array_start[i] == startup) {
        flag = 1;
        continue;
    }
    if (flag == 1){
        (__init_array_start[i])();
    }
}
```

Provided by @dangnhat

Chrono

- By now we have basic functionality on native
  - Start actors and send messages
  - But delayed messages never arrive
- Time is measured differently on RIOT
  - OS X/Linux use seconds since 1970-01-01
  - RIOT uses time since system start
- Most of the std::chrono is header only
  - We can include the header
  - Provide our own implementation
    - Timepoint class
    - Function to acquire the time
    - Breaks STL specification

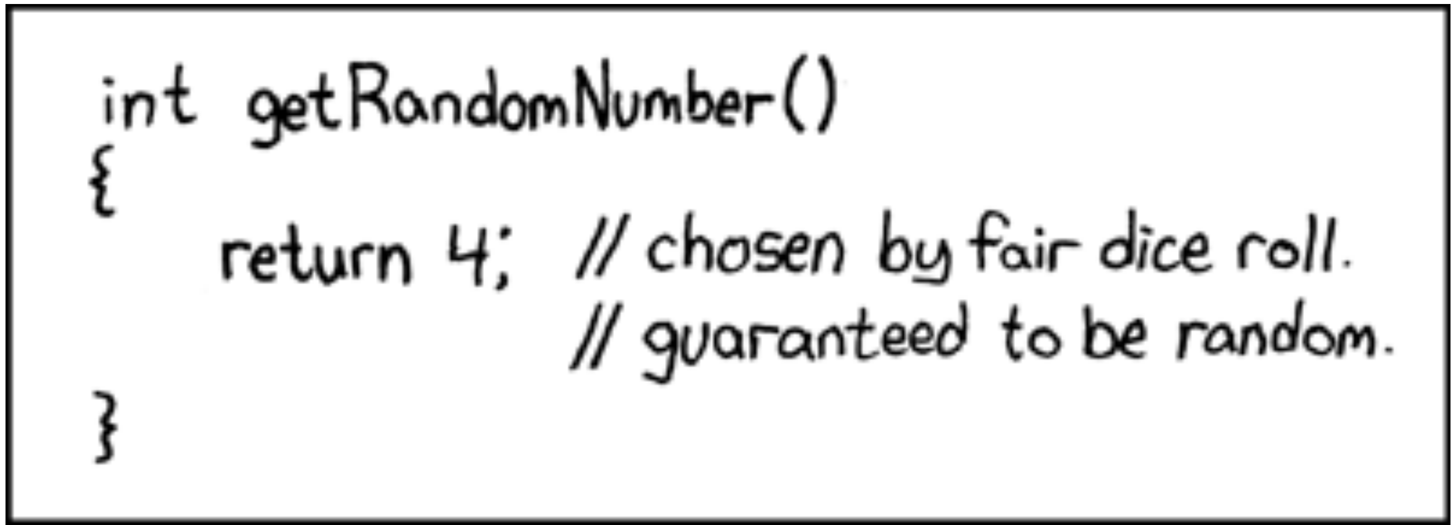
Demo Time!
(native)

CMake Cross compiling

- CMake supports toolchain files
  - -DCMAKE_TOOLCHAIN_FILE
  - Configure architecture, processor, compiler and flags
  - Created a file for the stm32f4discovery
- CMake automatically tests the compiler
  - Test fails when using the arm-none-eabi
  - Module CMakeForceCompiler should fix this
  - Did not work for me, can be achieved manually

Moving to arm-none-eabi

- Startup files handle static initialization
- libstdc++ for ARM is not complete
  - Can not provide hardware/OS dependent impl.
  - Does not include to_string
- Missing dso handle
  - Must be defined during startup to use global objects
- Actors use hardware address for their ID
  - stm32f4 does not have one, make it random



```
int getRandomNumber()
{
        return 4;  // chosen by fair dice roll.
                   // guaranteed to be random.
}
```

[1]

Embedded Debugging

- There is a GDB for arm-none-eabi
- CAF with debug symbols is huge
  - Only link specific objects with debug
- Files not found to show code position
  - Moving code to the “right” path helps
- Some breakpoints can not be set
- No backtrace

Where we are now

- Extended STL functionality on RIOT
  - Thread, mutex, condition variable, (chrono)
  - Needs to be turned into a PR
- Limited support for CAF on RIOT
  - On native port all my tests succeeded
  - On hardware some problems persist
  - Work on IO did not start yet

Demo Time!
(stm32f4discovery)

Agenda

1. General Background
2. Actors for the IoT
3. CAF on RIOT
4. **Experimentation**
5. Conclusion and Future Work

Exceptions

- Disabled in GCC for some architectures
    - Luckily not for the stm32f4discovery
- Exception cause the board to restart
- Requires memory specific region
    - Saved to eh_frame section
    - Found startup files only
    - Support for other boards in RIOT
- Did not work for the stm32f4discovery

Security

- Authentication, authorization and encryption
  - Establish encrypted channels (DTLS)
  - Generate key at local TA (key generation)
  - Authenticate runtime environments
- Challenges
  - Constrained power & energy
  - Nodes physically acquired
- Crypto is hard to do right

Test Environments

- Comfortable and fast vs. realistic and slow
- RIOT offers a native port
  - Not a realistic environment
- Few nodes in our lab
  - 7 Raspberry Pis running Linux
- FU Berlin (DES Testbed)
  - 60 nodes distributed in several rooms and floors
- INRIA Technology Institute in France
  - Connected through RIOT and Safest
  - 2700 nodes distributed through France

6LoWPAN USB Dongles

- IA-OEM-DAUB1
  - Drivers for Windows and Linux (only old kernels)
  - Not open, but include binary-blob
- atusb
  - Tip from the linux wpan IRC
    - Drivers not in mainline kernel (but netnext)
    - Merged our own kernel for the Raspberry Pi
  - The last one was delivered to us
  - Design is open, but expensive to produce only a few
- R-Idge
  - Suggested on the RIOT mailing list
  - Easy to use & available

Agenda

1. General Background
2. Actors for the IoT
3. CAF on RIOT
4. Experimentation
5. Conclusion and Future Work

Conclusion

- Took much longer than I expected
  - Finding the thread mistake took me ~1 ½ weeks
  - Spent a lot of time with the debugger
- Some mistakes could have been avoided
  - By a complete picture of the functionality
  - More test-cases (e.g., test-first)
- Will probably be faster next time (libcaf_io?)

Some Future Work

- Get this running on the stm32f4discovery
- Move threads, mutex, ... to RIOT
- Implement the network stack
- Port libcaf_io to RIOT
- Enable exceptions
- Include a security concept
- Do lots of testing

Thanks for Listening

Thanks to Martin and Dominik,
they helped a lot!

References

[1] xkcd, http://xkcd.com/221/, December 2014, under Creative Commons Attribution - NonCommercial 2.5

[2] D. Charousset, R. Hiesgen, and T. C. Schmidt, “CAF - The C++ Actor Framework for Scalable and Resource-efficient Applications,” in *Proc. of the 5th ACM SIGPLAN Conf. on Systems, Programming, and Applications (SPLASH ’14), Workshop AGERE!*, New York, NY, USA: ACM, Oct. 2014.

[3] C++ Actor Framework, “CAF”, http://actor-framework.org, December 2014.

[4] RIOT-OS., “RIOT,” www.riot-os.org, December 2014.

[5] INRIA, “FIT/IoT-LAB,” https://www.iot-lab.info, December 2014.