



# Common API C: Status And Performance

2016-04-28 | Developers

Pavel Konopelko  
Software Architect  
Visteon

This work is licensed under a Creative Commons Attribution-Share Alike 4.0 ([CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/))

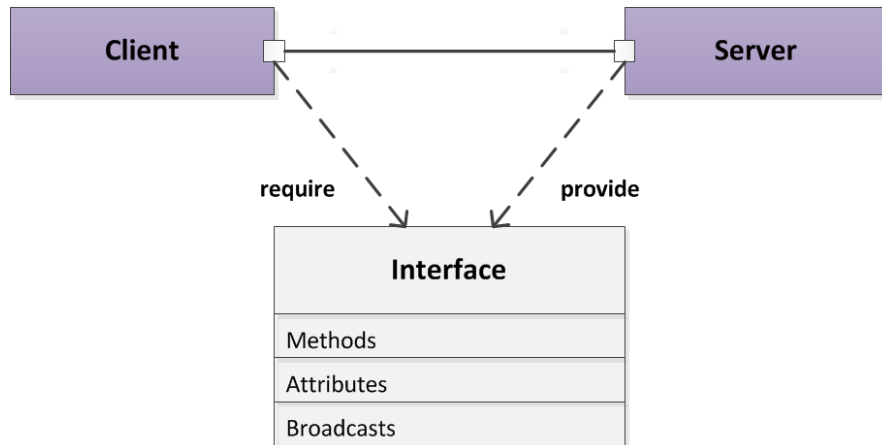
GENIVI is a registered trademark of the GENIVI Alliance in the USA and other countries

Copyright © GENIVI Alliance 2016

28-Apr-16

# INTRODUCTION

# Purpose of Common API



- Client and Server communicate via the Interface that the Server provides and the Client requires
- Interface is defined in Franca IDL and include methods, attributes and broadcasts
- Interface can have multiple instances that are identified by their names
- Interface representation in programming language is pre-defined and is generated automatically together with the communication backend code
- Client and Server logic is backend-independent and compatible with any supported backend

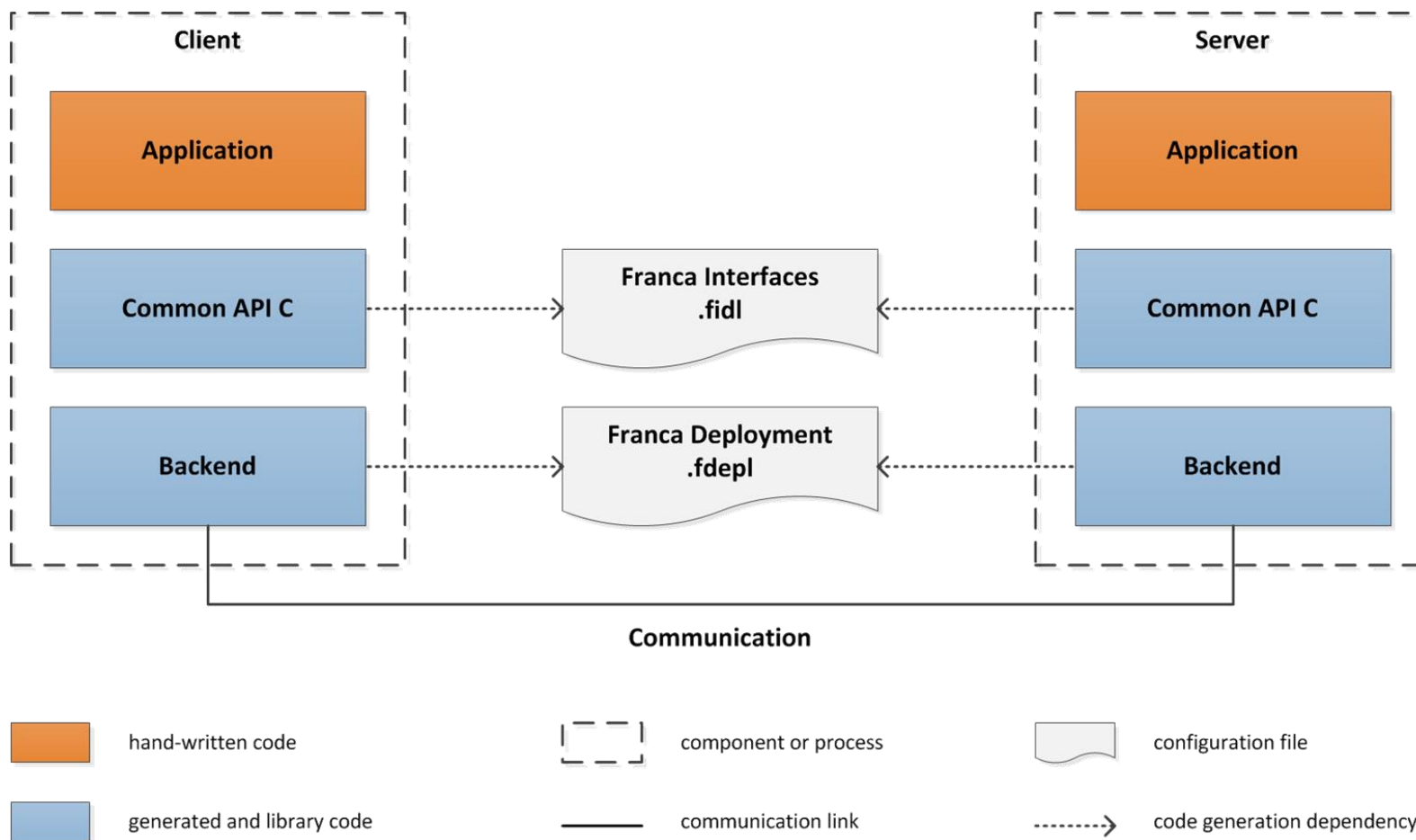
# Related Projects

Common API C is related to other projects supported by GENIVI:

- Franca provides common IDL and infrastructure for code generation
  - <http://franca.github.io/franca/>
- Common API C++ implements C++ bindings
  - <http://projects.genivi.org/commonapi>
- Yamaica supports transformations between Franca IDL and UML
  - <http://projects.genivi.org/yamaica/>

The Franca logo, with the word "Franca" in orange. The letter 'F' is stylized with a white circle inside its top loop.The CommonAPI C++ logo, with the text "CommonAPI C++" in a grey, metallic, 3D-style font.

# High-Level Architecture



# Example of Application Code

```
/* ---- Calculator.fidl ---- */
package org.test
interface Calculator {
  version { major 0 minor 1 }
  method add {
    in { Double left
        Double right }
    out { Double sum } }
}
/* ---- client.c ---- */
cc_backend_startup();
cc_client_Calculator_new(
  "org.test.Server:/calculator:org.test.Calculator",
  NULL, &calculator);
cc_Calculator_add(calculator, 3.1415, 2.7182, &sum);
calculator = cc_client_Calculator_free(calculator);
cc_backend_shutdown();
```

```
/* ---- server.c ---- */
static int Calculator_impl_add(
  struct cc_server_Calculator *instance,
  double left, double right, double *sum) {
  *sum = left + right;
  return 0;
}
static struct cc_server_Calculator_impl impl =
{ .add = &Calculator_impl_add };
/* ... */
cc_backend_startup();
cc_server_Calculator_new(
  "org.test.Server:/calculator:org.test.Calculator",
  &impl, NULL, &calculator);
/* run backend event loop */
calculator = cc_server_Calculator_free(calculator);
cc_backend_shutdown();
```

# Proof of Concept Development

- Project is run under the governance of GENIVI System Infrastructure EG
  - The project relies on the public GENIVI infrastructure (git, e-mail, wiki and bug tracker)
  - Compliance roadmap targets SC-P2 initially and SC-P1 once sufficiently mature
- Proof of Concept (PoC) is currently under development
  - The goal is to better understand the requirements and solution architecture
  - The PoC scope includes both the run-time libraries and the code generator
  - The PoC runtime code is licensed under MPL-2.0, code generators under EPL-1.0
  - The v0.2.1 was released in April 2016
- Requirements and design ideas are documented in parallel to the PoC
  - See <https://at.projects.genivi.org/wiki/display/PROJ/Common+API+C>
  - The results are reviewed and discussed in the SI EG to agree on the project target



# Project References

- Source code repository
  - <http://git.projects.genivi.org/?p=common-api/c-poc.git;a=summary>
- Mailing list
  - [genivi-ipc@lists.genivi.org](mailto:genivi-ipc@lists.genivi.org)
- Wiki home page
  - <https://at.projects.genivi.org/wiki/display/PROJ/Common+API+C>
- Bug tracker
  - <https://at.projects.genivi.org/jira/projects/CC/issues/?filter=allopenissues>
- *Beware: The repository will soon move to [github.com/genivi](https://github.com/genivi)*



# STATUS UPDATE

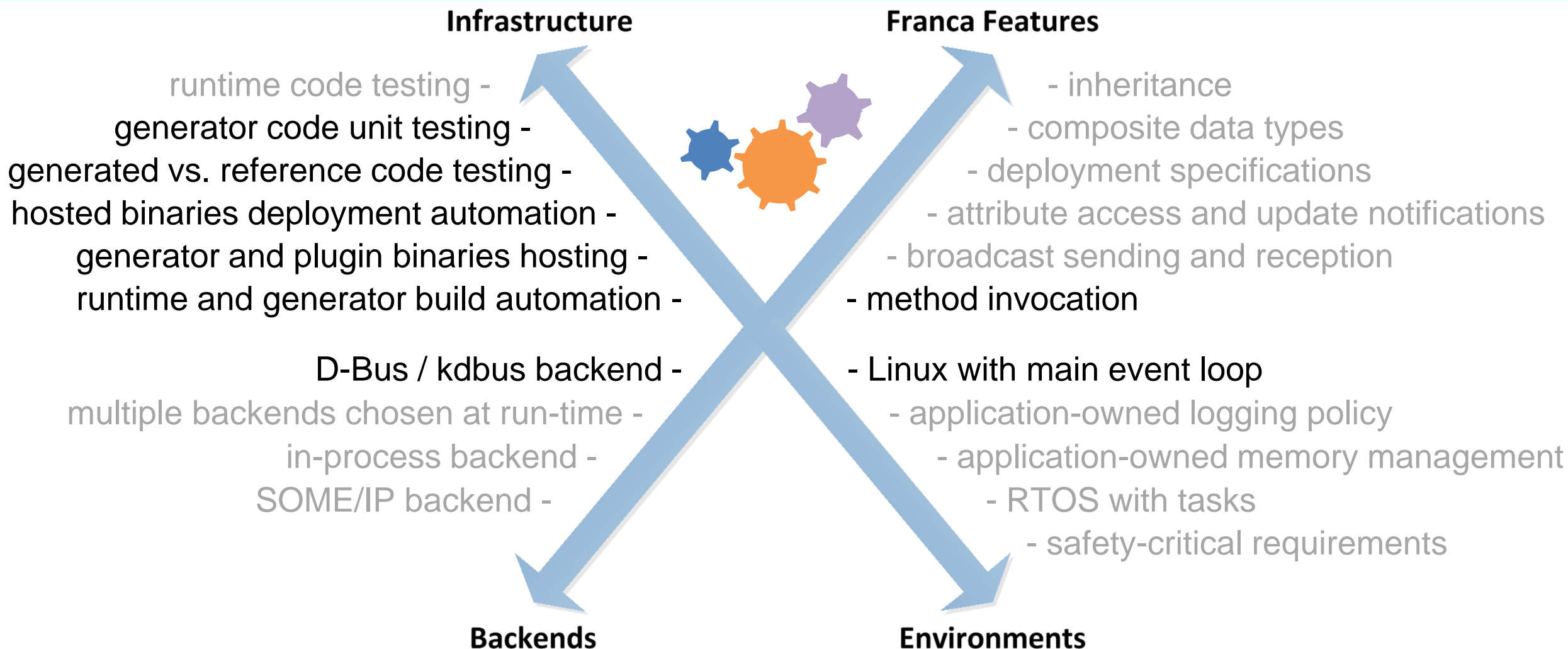
# PoC Development Progress

- Development approach is based on iterations
  - Manually develop application that uses certain Common API features (see below)
  - Incrementally implement corresponding aspects of the Common API C design
  - Extract shared implementation into a library and use the rest as generator test cases
- Functionality increments
  - Multiple interface instances and multiple interface implementations (DONE)
  - Asynchronous method invocations and backend event loop embedding (DONE, v0.1.0)
  - Code generation for currently supported features (DONE, v0.2.0)
  - Performance tests; improvements for build scripts and D-Bus backend (DONE, v0.2.1)
  - TBD: Support for Franca signals and attributes
  - TBD: Backend for in-process communication
  - TBD: Memory allocation managed by the application
  - TBD: Full support for Franca type system

# Dependencies as of Release v0.2.1

- Dependencies
  - systemd v219+ (versions before v221 with workarounds)
  - Eclipse Mars (tested on Mars.2)
  - JRE 1.7+ (release v0.2.x uses JRE 1.7)
  - Franca 0.10
- Self-hosted development
  - Fedora 23+
  - Ubuntu 15.10+
- Cross-development
  - Yocto 1.8/fido+

# Features And Roadmap



# PERFORMANCE BENCHMARKING

# Benchmark Design

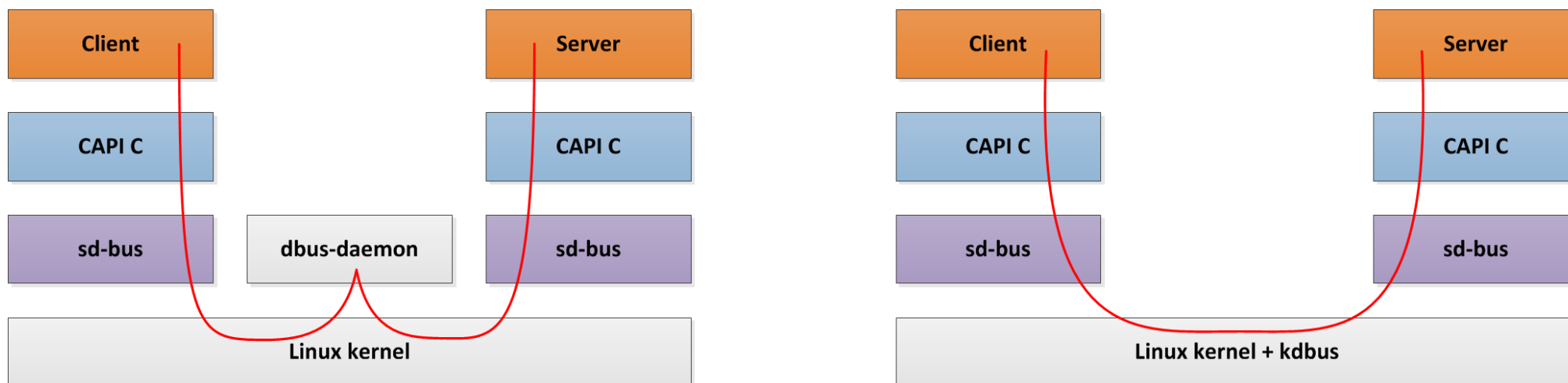
- Goals
  - Measure the overhead added by Common API C for the client synchronously invoking methods on the server
  - Compare the performance overhead to that added by Common API C++ for the same communication scenario
  - Compare the efficiency of libdbus vs. sd-bus in configurations with and without kdbus
- Design
  - Client invokes a method; server replies; client waits for the reply before invoking the method again
  - Compare invoking methods without arguments and with 40 bytes in and 40 bytes out arguments (Int32, 4 x Double, UInt32; server copies in values into out values)
  - Measure the total throughput on 100.000 method invocations

# kdbus Disclaimer

- kdbus was supported by systemd developers and was anticipated by many as a more performant replacement for dbus-daemon
- Several attempts in 2014 and 2015 to upstream it, though, triggered lots of debates on LKML, but brought no results
- No public out-of-tree patch sets seem to be currently maintained; patches for the kernels 4.0 through 4.3 were not maintained since September 2015
  - see [github.com/systemd/kdbus](https://github.com/systemd/kdbus)
- systemd dropped some kdbus support (e.g., bus-proxyd) in early 2016
- Former kdbus developers seem to be working on something new
  - see [github.com/bus1](https://github.com/bus1)

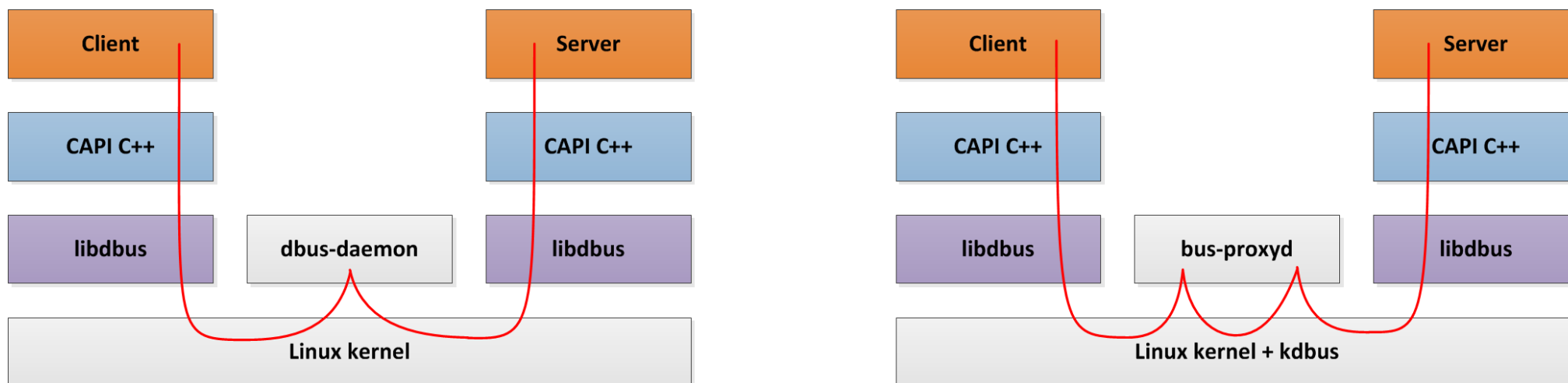


# Common API C Message Flow



- Using systemd sd-bus provides efficient binding for D-Bus messaging
- Using kdbus reduces the number of user-kernel context switches

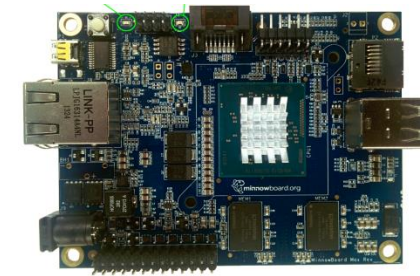
# Common API C++ Message Flow



- Using libdbus provides less efficient binding for D-Bus messaging
- Using kdbus increases the number of user-kernel context switches due to systemd bus-proxyd

# Test System Hardware

- ARM architecture:  
Renesas Porter Board
  - Renesas R-Car M2
  - x2 ARM Cortex-A15 @ 1.5GHz
  - 2GB DDR3-1660 RAM
- Intel x86 architecture:  
Intel Minnow Board MAX
  - Intel Atom E3825
  - x2 cores @ 1.33GHz
  - 2GB DDR3-1066 RAM



Images courtesy of <http://www.elinux.org>

# Test System Software

- Distro
  - Based on Yocto 2.0.1 / Poky jethro, core-image-minimal
  - Linux kernel 3.10.31 on ARM / 4.1.8 on x86
  - GCC 4.9.3 on ARM / 5.2.0 on x86; optimizations "-O2"
  - glibc 2.22
  - systemd v225
  - D-Bus 1.8.20 plus patches from CAPI C++ 3.1.5p2
  - kdbus from [github.com/systemd/kdbus](https://github.com/systemd/kdbus), branch v4.1, latest
- Benchmark
  - capic v0.2.1; including test/perf and test/capic++-perf
  - Common API C++ 3.1.5p2

# Test Execution

- 3 system configurations
  - ARM, x86, x86 + kdbus
- 8 benchmarks
  - sd-bus, capic, libdbus, capic++
  - without and with ("-p") arguments
- For each benchmark
  - Boot and connect via serial line
  - Launch server
  - 12 times launch client to run a test sequence with 100,000 messages
  - Shutdown

Poky (Yocto Project Reference Distro) 2.0.1 intel-corei7-64 ttyS0

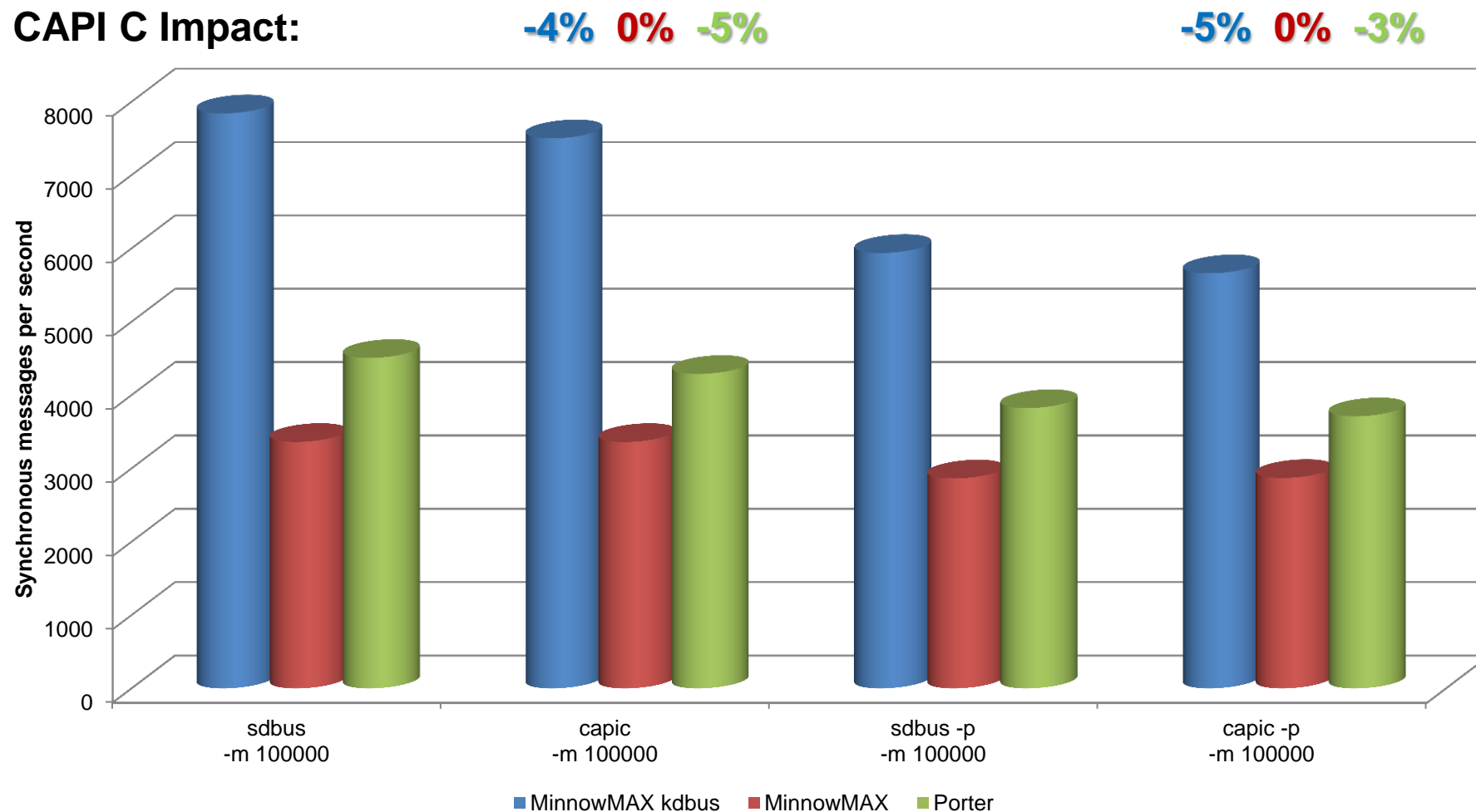
```
intel-corei7-64 login: root
root@intel-corei7-64:~# sdbus-server &
root@intel-corei7-64:~# Started sdbus-server
entering main loop...
root@intel-corei7-64:~# for n in `seq 12`; do sdbus-client -m 100000; done
Started sdbus-client
starting test...
test completed
message payload [bytes]: 0
sync messages sent:      100000
messages per [s]:       3309.13
exiting sdbus-client
```

...

```
Started sdbus-client
starting test...
test completed
message payload [bytes]: 0
sync messages sent:      100000
messages per [s]:       3241
exiting sdbus-client
root@intel-corei7-64:~# reboot
```

# Common API C with sd-bus

## CAPI C Impact:

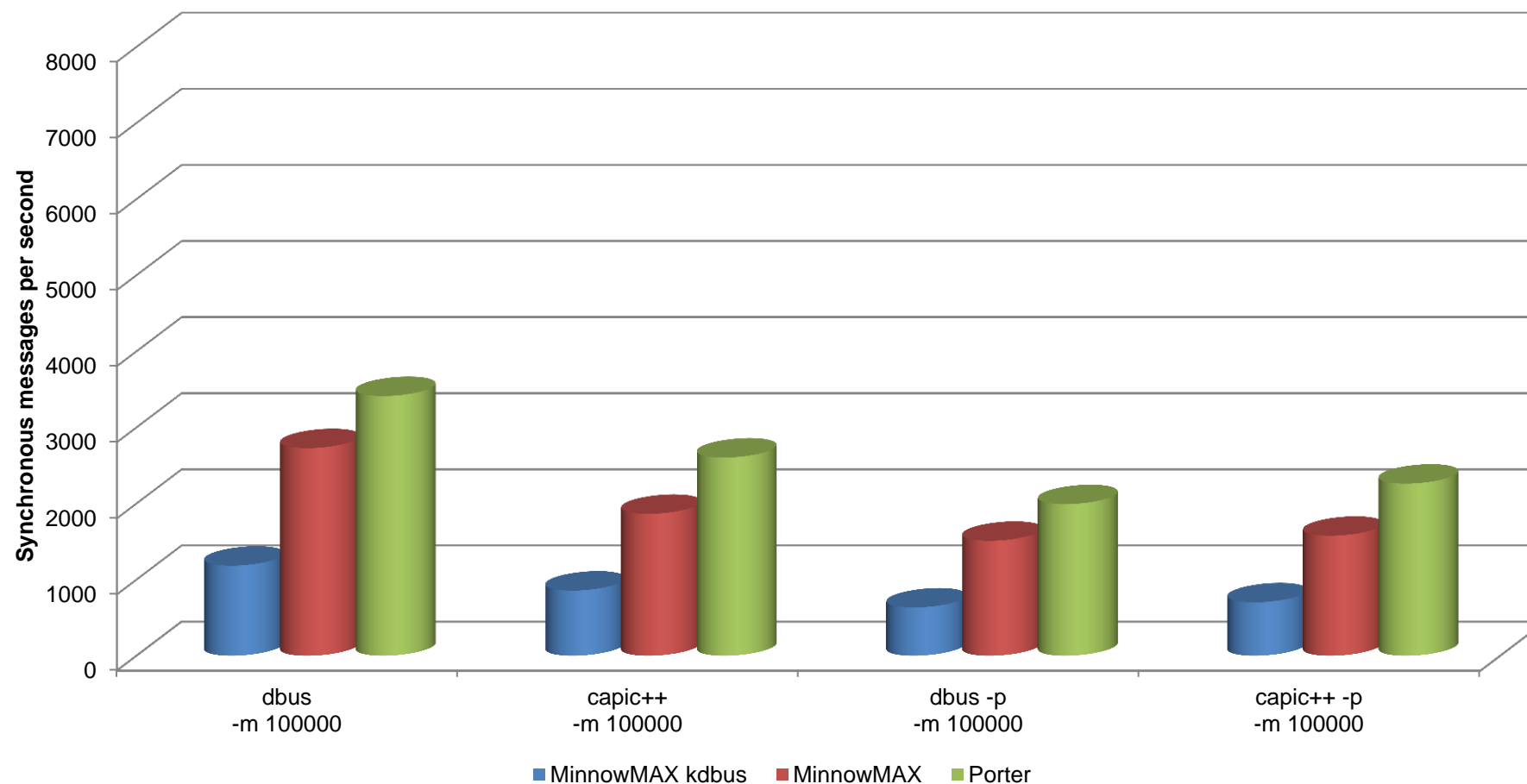


# Common API C++ with libdbus

**CAPI C++ Impact:**

**-28% -32% -24%**

**+11% +4% +14%**





# BONUS TRACK

# Code Generation as a Build Step

```
# Excerpt from perf/Makefile.am

bin_PROGRAMS = capic-client capic-server

capic_client_SOURCES = \
    src/capic-client.c
nodist_capic_client_SOURCES = \
    src-gen/client-TestPerf.c \
    src-gen/client-TestPerf.h

capic_server_SOURCES = \
    src/capic-server.c
nodist_capic_server_SOURCES = \
    src-gen/server-TestPerf.c \
    src-gen/server-TestPerf.h

BUILT_SOURCES = \
    src-gen/client-TestPerf.h \
    src-gen/server-TestPerf.h

CLEANFILES = src-gen/*.c src-gen/*.h

# capic-core-gen requires absolute filename as its argument
src-gen/client-%.c src-gen/client-%.h \
src-gen/server-%.c src-gen/server-%.h: %.fidl
    arg=$(basename $<) ; capic-core-gen $(abs_srcdir)/$${arg}
```

```
# Excerpt from capic++-perf/CMakeLists.txt

set(FIDL_DIR ${CMAKE_SOURCE_DIR}/fidl)
set(SRC_GEN_DIR ${CMAKE_BINARY_DIR}/src-gen)
set(PERFTEST_DIR ${SRC_GEN_DIR}/v0/org/genivi/capic)

set(CAPICXX_CORE_GEN_CMD capic++-core-gen -sk)
set(CAPICXX_DBUS_GEN_CMD capic++-dbus-gen)

set(PERFTEST_CORE_OUTPUT
    ${PERFTEST_DIR}/TestPerf.hpp
    ${PERFTEST_DIR}/TestPerfProxyBase.hpp
    ${PERFTEST_DIR}/TestPerfProxy.hpp
    ${PERFTEST_DIR}/TestPerfStubDefault.cpp
    ${PERFTEST_DIR}/TestPerfStubDefault.hpp
    ${PERFTEST_DIR}/TestPerfStub.hpp)
add_custom_command(
    OUTPUT ${PERFTEST_CORE_OUTPUT}
    COMMAND ${CAPICXX_CORE_GEN_CMD} ${FIDL_DIR}/TestPerf.fidl
    DEPENDS ${FIDL_DIR}/TestPerf.fidl
    WORKING_DIRECTORY ${CMAKE_BINARY_DIR})
# D-Bus code generation is done in a similar way

add_executable(capic++-client
    src/capic++-client.cpp
    ${PERFTEST_DIR}/TestPerf.hpp
    ${PERFTEST_DIR}/TestPerfDBusProxy.cpp
    ${PERFTEST_DIR}/TestPerfDBusDeployment.cpp)
```

# Code Generation as bitbake Recipes

```
# Excerpt from capic-core-native_0.2.1.bb

BASE_URL = "http://docs.projects.genivi.org/common-api-c"
BASE_VER = "${@'.'.join(d.getVar('PV', True).split('.')[0:2])}"
SRC_URI = "${BASE_URL}/generator/${BASE_VER}/${PV}/capic-core-gen.zip"

def get_launcher_name(d):
    BS = d.getVar('BUILD_SYS', True)
    if BS == "x86_64-linux":
        launcherName = "capic-core-gen-linux-x86_64"
    elif BS == "i686-linux":
        launcherName = "capic-core-gen-linux-x86"
    return launcherName

inherit native

S = "${WORKDIR}"
DD = "${D}${datadir}/${PN}-${PV}/"
LAUNCHER = "${@get_launcher_name(d)}"

do_install() {
    install -d ${DD}
    cp -f ${S}/artifacts.xml ${DD}
    # ... repeat for all other generator components
    install -d "${D}${bindir}/"
    ln -sf ${DD}${LAUNCHER} ${D}${bindir}/capic-core-gen
}
```

```
# Excerpt from capic-perf_0.2.1.bb

SRC_REPO = "git://git.projects.genivi.org/common-api/c-poc.git"
SRC_URI = "${SRC_REPO};branch=master;tag=${PV}"
S = "${WORKDIR}/git"

DEPENDS = "systemd capic-core capic-core-native"

inherit autotools pkgconfig



---



# Excerpt from capic++-perf_0.2.1.bb

SRC_REPO = "git://git.projects.genivi.org/common-api/c-poc.git"
SRC_URI = "${SRC_REPO};branch=master;tag=${PV}"
S = "${WORKDIR}/git"

DEPENDS = " \
    capic++-core capic++-core-native capic++-dbus capic++-dbus-native"

inherit cmake pkgconfig
```