Cross-Platform Protocol Development Based on OMNeT++

Stefan Unterschütz, Andreas Weigel and Volker Turau

SIMUTools 2012: OMNeT++ Workshop

March 23rd, 2012

Institute of Telematics
Hamburg University of Technology

TUHH

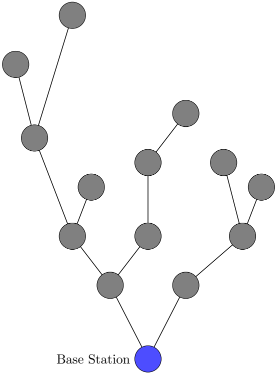
1

Introduction

Motivation

Simulation is indispensable for the development of (wireless) network protocols.

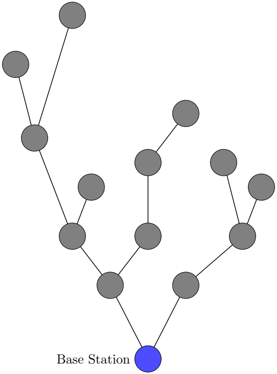
OMNeT++ is a powerful tool for simulations of network protocols.



Motivation

Simulation is indispensable for the development of (wireless) network protocols.

OMNeT++ is a powerful tool for simulations of network protocols.



However:

Re-implementation of protocols for a target platform is time-consuming and error-prone

Introduction

CometOS,
a component-based, extensible, tiny
“operating system”

Design Goals

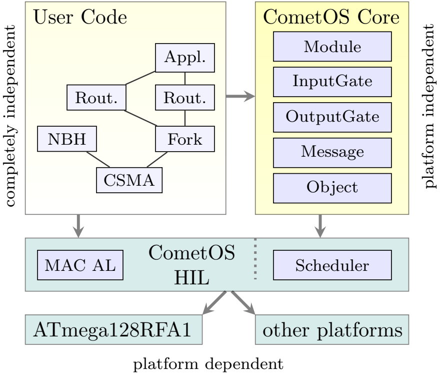
- Single code base for protocols, whether running simulations or executing on target hardware
- “Lightweight enough” for resource constrained hardware
- Flexibility, extensibility, avoidance of code redundancy
- Thereby: speed up protocol development and produce safe code

1. Introduction
2. Architecture and Concepts
3. Feasibility
4. Conclusion

2

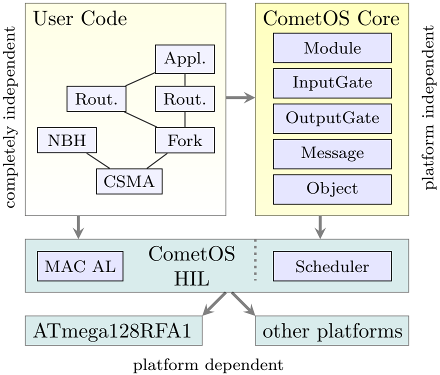
Architecture and Concepts

Architecture

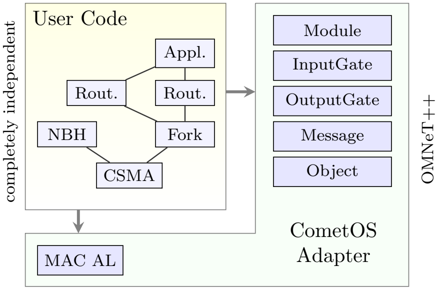


Hardware platform

Architecture



Hardware platform



OMNeT++ simulation

Gates and Message Passing

Message handlers are executed non-preemptively
(millisecond precision)

- Adoption of OMNeT++ message and gate concept
- Added type safety
  - Gates instantiated with a certain message type
  - Connections between gates are checked at compile time
  - ⇒ **`dynamic_cast`**s can be avoided
- Decrease of boilerplate code
  - Gates and self-messages directly bound to handler methods
  - No `handleMessage()` dispatch code necessary
- User-defined messages
  - Created by deriving from base class
  - Basic message types provided: Request/Confirm, Indication

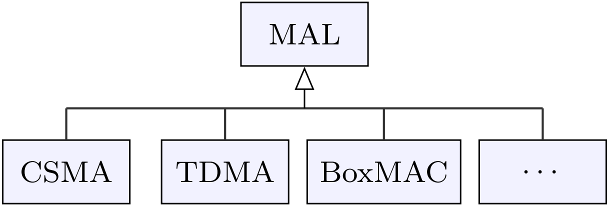
Message Passing (2)

```
class MyMsg: public Message {};

class MyReceiver:
public Module {
public:
  InputGate<MyMsg> gateIn;

  MyReceiver() :
    gateIn(this,
           &MyReceiver::handle,
           "gateIn")
  {}

  void handle(MyMsg *msg) {
    delete msg;
  }
};
```

```
class MySender:
public Module {
public:
  OutputGate<MyMsg> gateOut;

  MySender() :
    gateOut(this, "gateOut")
  {}
  void initialize() {
    schedule(new Message,
      &MySender::traffic,500);
  }
  void traffic(Message *msg) {
    gateOut.send(new MyMsg);
    delete msg;
  }
};
```

MAC abstraction layer

- Goal: Basis for arbitrary, **platform-independent** MAC protocols (CSMA, TDMA, LPL, LPP)
- Should support Link-Layer ACKs, CCA, Random Backoffs
- Hardware-supported functions of 802.15.4 transceivers



Airframes and Serialization

- Actual over-the-air packet: Managed byte array (Airframe)
- Support for serialization of simple types
- User-defined types (structs, classes):
  - ⇒ serialization user-provided

```
struct NwkHeader {
  uint16_t dst;
  uint16_t source;
}
void serialize(ByteVector &buffer, const NwkHeader &value) {
  serialize(buffer, value.dst);
  serialize(buffer, value.source);
}
...
NwkHeader nwk(SINK_ADDR, getId());
request->getAirframe().serialize(nwk);
```

Initialization

For OMNeT++

⇒ .ned, .ini files

```
// Setup for OMNeT++ in NED language
// (skipped declaration of modules)
network Network {
    submodules:
        s: MySender;
        r: MyReceiver;
    connections:
    s.gateOut --> r.gateIn;
}
```

For Hardware Platforms:

⇒ C++ initialization file

```
// Setup for Hardware
MySender s;
MyReceiver r;

int main() {
  s.gateOut.connectTo(r.gateIn);
  cometos::initialize();
  cometos::run();
  return 0;
}
```

Base Station Support

- Currently under development
- Python wrapper for existing CometOS C++ code (SWIG)
  - Reuse protocol implementation for a base station
  - Usable with real testbed or OMNeT++ real-time simulation and TCP/IP connector
- Integration of powerful remote access methodology
  - Read/write of variables
  - Remote execution of methods
  - Subscribe to events

Base Station Support

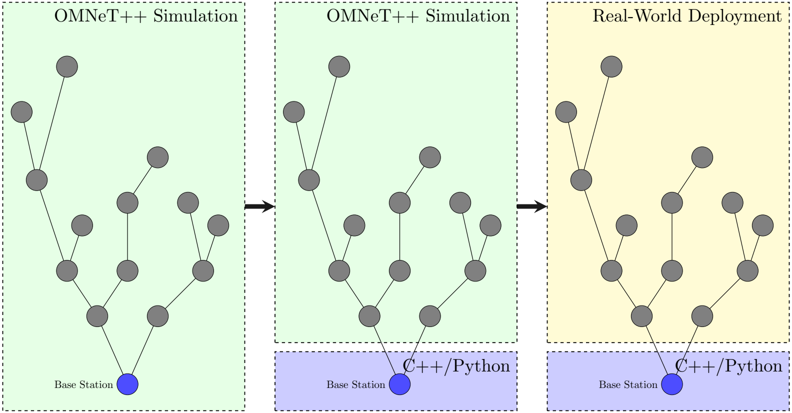
```
class MyModule :
public RemoteModule {
public:
  MyModule(const char* name) :
    RemoteModule(name) {}

  void initialize() {
    declareRemote(&MyModule::add,
                  "add");
  }
  uint16_t add(uint8_t &a,
               uint8_t &b) {
    return a+b;
  }
};
MyModule m("myModule");
```

```
r=RemoteModule("myModule");
r.declareRemote("add",
         uint16_t,
         uint8_t,
         uint8_t)
print r.add(18, 11)
>>> 29
```

↑ Python console

← CometOS-Module

Typical Development Steps



3

Feasibility

Resource Demand

- Minimum example (MySender, MyReceiver)

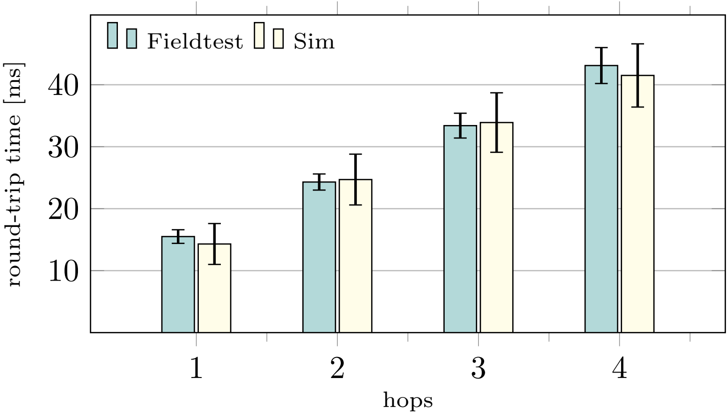
MCU	Flash	RAM
ATmega128RFA1	4148 bytes	145 bytes
LPC1763	3136 bytes	120 bytes

- 7 modules, forked protocol stack

MCU	Flash	RAM
ATmega128RFA1	10 kB	649 bytes
LPC1763	7 kB	580 bytes

Simulation Accuracy

- Comparison of RTTs from field installation (93 nodes at heliostat power plant in Jülich) and simulation for different number of hops



4

Conclusion

Conclusion, Future Work

- CometOS meets its design goals
  - Protocol implementations reusable on target hardware
  - “Lightweight enough”
- Field test at heliostat power plant in Jülich, Germany successfully running since May 2011
- Current and Future Work:
  - Smart Metering application based on CometOS
  - Improvement and extension of interface to driver layer
  - Direct support for logging and statistics recording and reporting

http://www.ti5.tu-harburg.de/research/projects/cometos/

Cross-Platform Protocol Development Based on OMNeT++

Stefan Unterschütz, Andreas Weigel and Volker Turau

SIMUTool

Andreas Weigel
Research Assistant

Phone +49 / (0)40 428 78 3746
e-Mail andreas.weigel@tu-harburg.de

http://www.ti5.tu-harburg.de/staff/weigel

Institute of Telematics
Hamburg University of Technology
TUHH

Resource Demand Revisited

- RAM usage depends on target architecture (e.g., 8 bit vs 32 bit)
- Values for 32 bit MCU
  - Module: 8 Bytes
  - InputGate: 16 Byte
  - OutputGate: 4 Byte
  - RemoteModule: 30 Bytes (including Module)
  - Standard modules Layer and Endpoint with 4 and 2 Gates require 70 bytes and 50 bytes
- ROM usage even more depends on architecture, instruction set, compiler etc.

Experiment Setup

- Packets with 50 bytes payload
- 100 measurements per node
- 802.15.4 (2.4 GHz ISM band, 250 kbps)

Cross-Layer Support

Communication between non-adjacent modules?

Cross-Layer Support

Communication between non-adjacent modules?

- Similar to OMNeT++'s `ControlInfo` or ns3's object aggregation:
  - Attach arbitrary objects to Messages and Airframes
- Example: Setting MAC txPower from higher layer:

```
// Application: set tx power to -20 dBm
request->add(new MacTxPower(-20));
...
// MAC: use MacTxPower if set
MacTxPower* txPower= request->get<MacTxPower>();
if (txPower != NULL) {...}
```