TCP for OMNeT++

Roland Bless
Mark Doll

Institute of Telematics
University of Karlsruhe, Germany

Overview

- Motivation
- Introduction OMNeT++ & TCP
- Concept for integration
- Implementation problems
- Evaluation results
- Summary & Outlook

Motivation I

- Investigation of Network Protocols
  - Different parameter settings
  - Extreme conditions
  - Scaling properties
  - → Difficult in testbeds
  - → Simulation
- Expressive simulation results
  - Require good emulation of real world behavior
  - Protocols we design use or modify TCP/IP stack
  - Emulation of network oriented layers
    - Transport (TCP)
    - Internet (IP)
    - Network Access (Ethernet)
  - → Need TCP/IP stack implementation for simulation

Motivation II

- Problem
  - no validated (tested to be compliant to the standard) TCP/IP implementation for OMNeT++
- Possible solutions
  - Choose other simulator
    - ns-2
    - OPNET
  - New implementation
  - Revision of the existing implementation (i. e. from Communications Engineering Institute, Karlsruhe)
    - full featured?
    - validated?
  - Re-use an existing real world implementation
    - Linux
    - FreeBSD

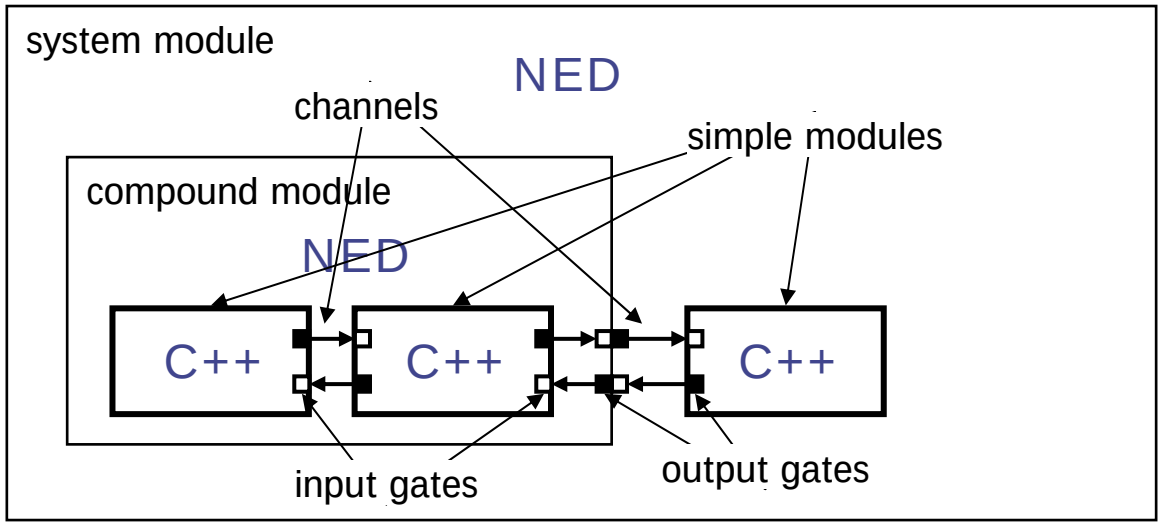
OMNeT++

- Discrete event simulator
  - hierarchically nested modules
  - communicate with messages through channels
- Written in C++
  - Complete source code publicly available
- Free for academic use
  - Commercial version OMNEST™
- Advantages
  - Very well structured
  - Highly modular
  - Not limited to network protocol simulations (i. e. like ns-2)
- Disadvantages
  - Relatively young: 1997 first public release

  → Few simulation models

OMNeT++

- Hierarchical nested modules



- *system* module
  - top level module
- *simple* modules
  - C++
  - NED description of interface: parameters and gates
- *compound* modules
  - NED only: parameters, gates and connections
  - unlimited nesting

Transmission Control Protocol

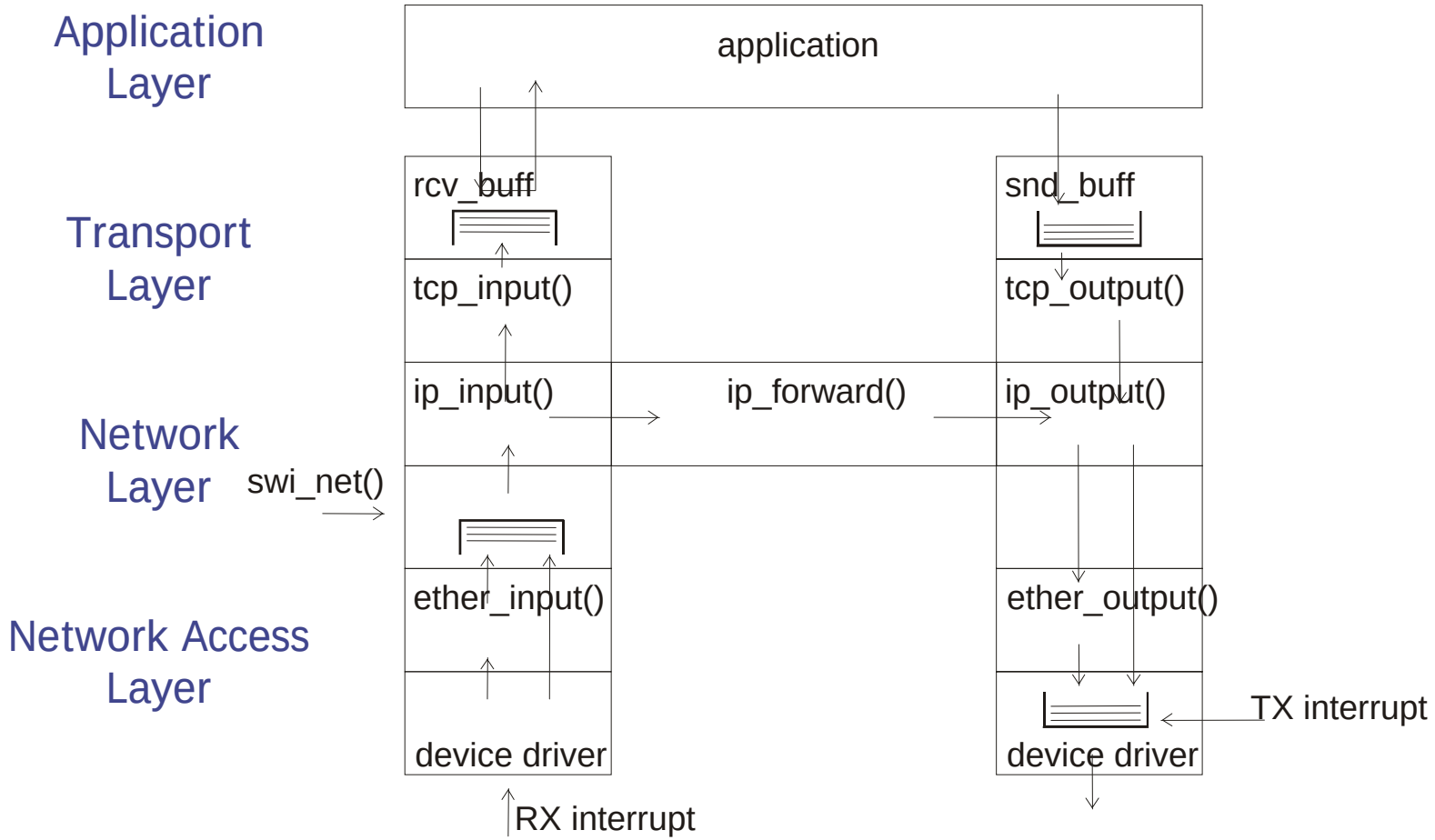
- TCP
  - Transport layer protocol (layer 4)
  - connection-oriented, reliable, stream-oriented
  - flow control, congestion control
    - Influenced by round trip time, packet loss
  - Most of today's internet traffic via TCP
    - TCP behavior influences behavior of many protocols & applications
  - API: BSD socket interface (socket type: stream)
- FreeBSD's TCP/IP implementation
  - source code freely available
  - well structured code (not as optimized as Linux)
  - 4.x BSD was starting point for many other implementations
    - MAC OS X essentially *is* FreeBSD (with additions like Aqua)
  - Network research was and still is based on (Free)BSD
    - Mobile IPv6, Protocol Independent Multicast, ...

FreeBSD vs. OMNeT++

- Function Call
- One TCP stack per host
- Multi-tasking, threads, functions are interruptible

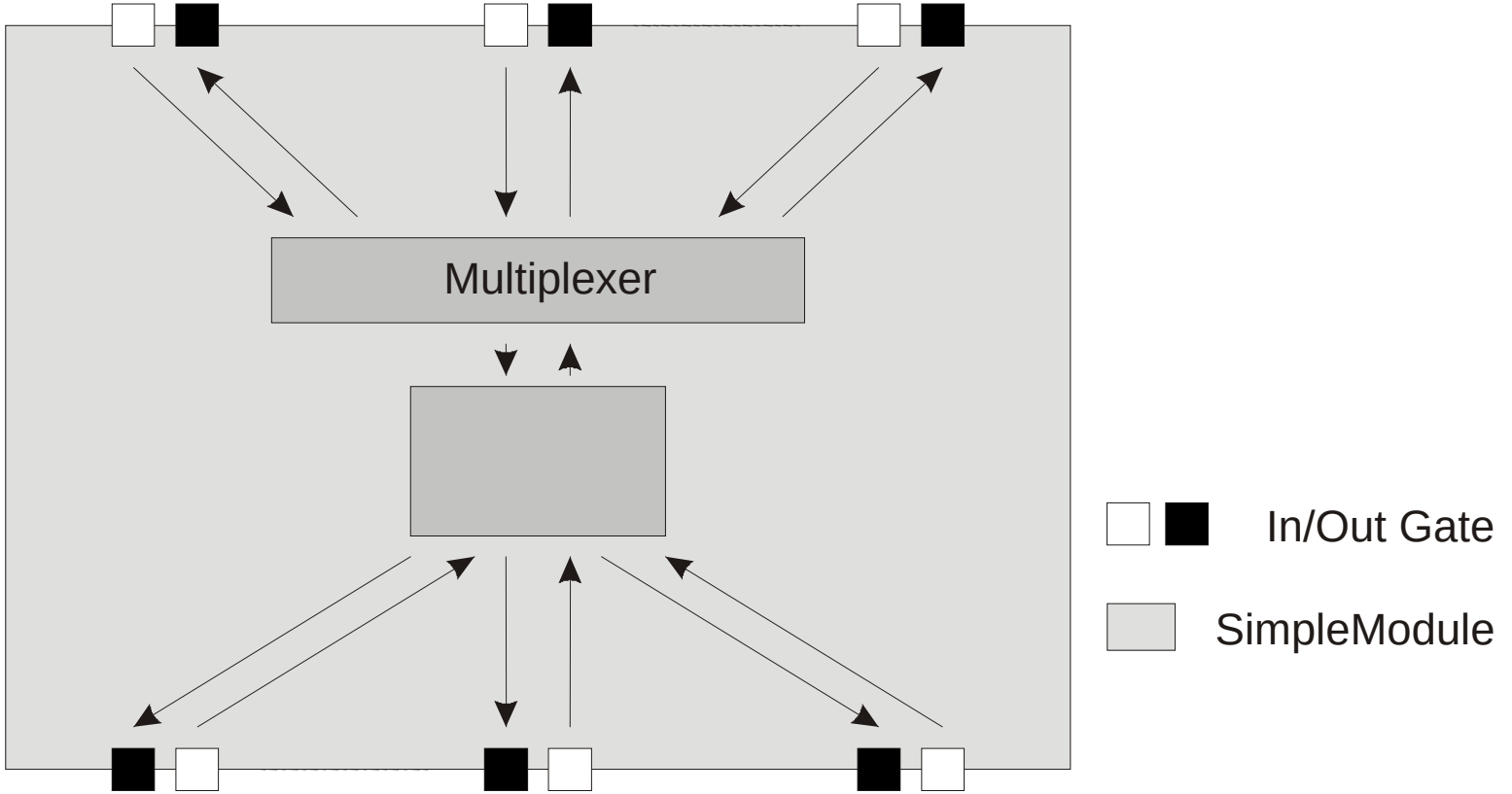
- Messages
- Many TCP stacks per simulation
- Messages must be processed in one pass

Protocol stack of FreeBSD



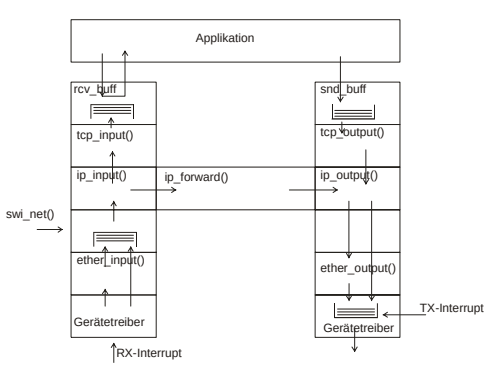
cHost concept I

Gates to Applications (cAppl)

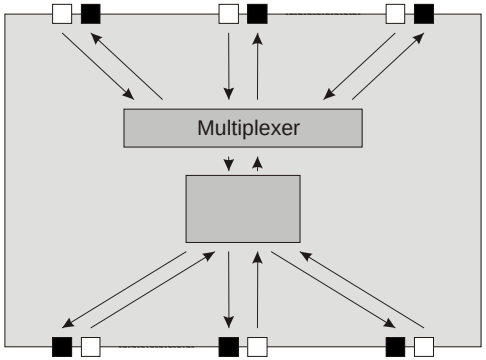


Gates to LANs or other Hosts (cMedium or cHost)

cHost concept II

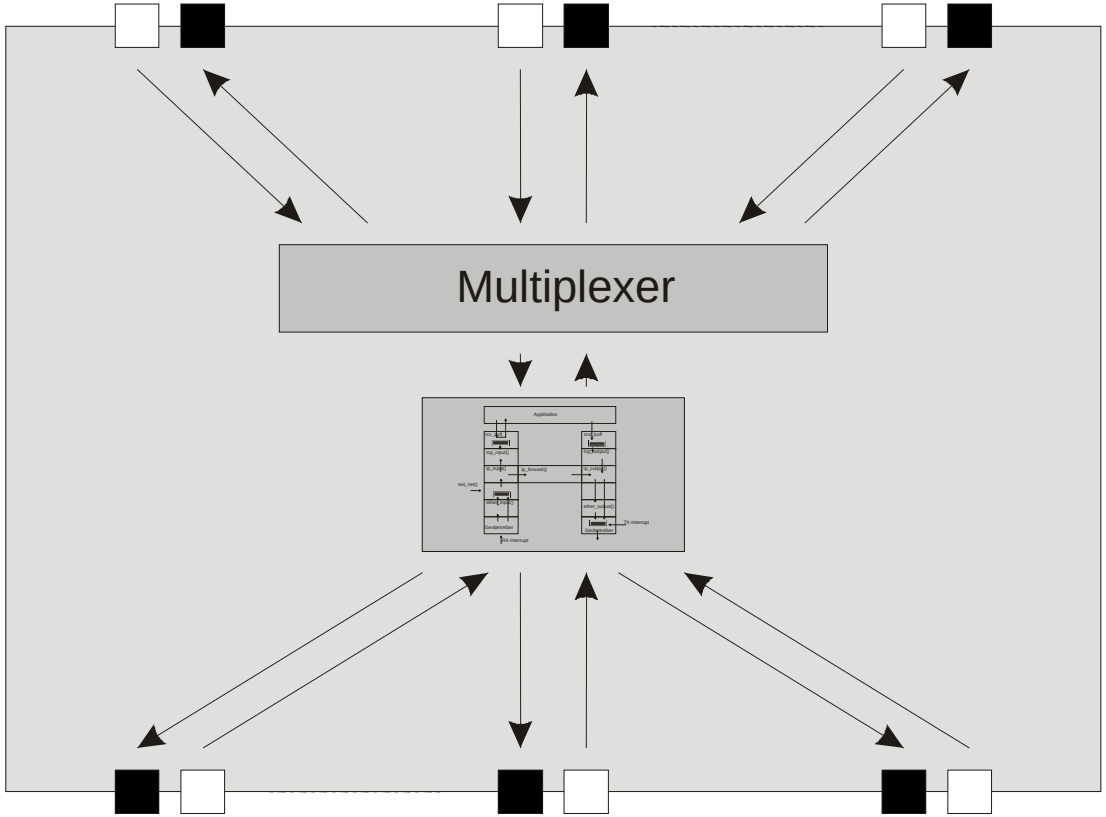


+

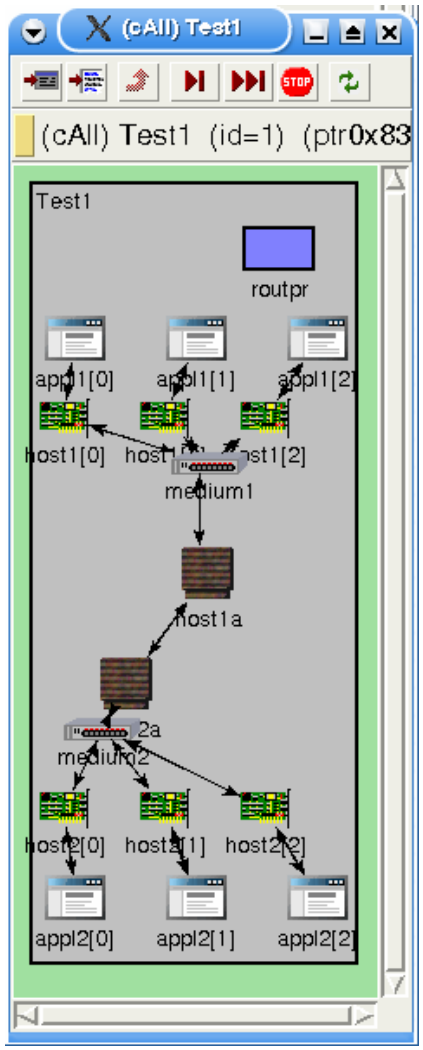


= ?

cHost concept III



OMNeT++ classes



- cHost
  - Simulated host with TCP/IP
- cMedium
  - Emulates broadcast medium (Ethernet LAN)
  - Allows for cHost address autoconfiguration
- cAppl
  - Source/sink (sample application)
- cRoute
  - Generates routing tables

cAppl & cRoute

cAppl

- Uses socket interface to cHost
  - Demonstrates usage of message based socket interface
- Can accept and initiate connections
- Full duplex

cRoute

- Manual routing table set up
  - Time consuming
  - Configuration Errors
  - Often routing table details unimportant
- Use cRoute class to generate routing tables for FreeBSD
  - Utilizes standard OMNeT++ class cTopology
  - Computes shortest path (Dijkstra)

cHost <-> FreeBSD I

- TCP stacks must be independent
- No global and static variables possible
- Access via C macro

```
#define D    ( (struct private_data*)bsd_data)
```

  - Usage: replace `xyz` with `D->xyz`
- FreeBSD: same identifier for types and variables
  - i. e. `ifnet`
  - Simple substitution of `xyz` with macro impossible
  - However changes to FreeBSD source minimal
    - Avoid bugs
    - Allow for easier re-porting of future FreeBSD releases

cHost <-> FreeBSD II

- Switch to BSD via C macro (simplified)

```
#define ENTER_BSD() {       \
  bsd_data=data;            \
  host_class=this;          \
  host_id=id();             \
}
```

- Calling BSD functions from OMNeT++
  - Enclose call by `ENTER_BSD` and `LEAVE_BSD`
- Calling OMNeT++ functions from BSD
  - OMNeT++ function exported with C calling conventions
  - Function then uses `host_class` to access instance data

Technical Problems

- Include files
  - FreeBSD and OMNeT++ (host OS) need their own include files
  - i. e. `struct sockaddr` and `struct sockaddr_in`
- Socket interface
  - Function calls replaced by OMNeT++ messages
    - `read(), write(), listen(), connect(), ...`
  - Blocking functions emulated by non-blocking + self message
- Memory (shared)
  - Differences between libc of FreeBSD and host OS running OMNeT++
  - mbufs & mclusters mapped on malloc
- Timers
  - Access to BSD `ticks` variable redirected to `gettick_toomnet()` which uses OMNeT++'s `simtime()`
  - `startup` added individually per cHost to prevent synchronization
  - One timer for all hosts where appropriate, i. e. `ip_slowtimo`

Evaluation

- Memory usage per Host
  - FreeBSD: 19 KiB
  - + cHost class
  - + Receive-/Send buffer 64 KiB (default) per connection
- Run time in seconds for 1 hour of simulated time

Hosts/Conn.	0	1	2
10	0.467	2.199	4.196
100	3.361	30.575	59.638
1000	64.233	434.724	823.019

  - Time consuming insertion/removal of timer events in OMNeT++'s event queue (heap)

Summary

- First validated and complete TCP implementation for OMNeT++
- Realized by adapting FreeBSD‘s TCP/IP stack
- Message-based implementation of the BSD socket interface
- Routing module automates routing table setup

Outlook

- Performance
  - hide certain messages in OMNeT++
  - dedicated timer module for OMNeT++
- Improve adaption process
  - Perl script/parser for (semi)automatic global variable replacement
- Support further protocols
  - IPv6
  - System call interface for routing deamons
  - Mobile IPv6 patches

Thanks to Jérôme Freilinger
(diploma student)

Thank You for your attention!