Karlsruhe Institute of Technology

Integration of a GIST implementation into OMNeT++

Roland Bless and <u>Martin Röhricht</u>

Institute of Telematics, Department of Computer Science

Evaluation of Network Protocols

- Simulation
  - Investigation of scalability aspects
  - Study general behavior while varying parameters
- Implementation
  - Allows for performance investigations
  - Consideration of real-world aspects
- Possible way of validating protocols
  - Prototypical implementation for network simulator
  - Evaluation of implementation by simulations
  - Port implementation to work on real hardware

Integration of an existing implementation into OMNeT++

- Allow for protocol evaluations in large-scale scenarios

- Evaluation of existing implementation instead of model

- NSIS-ka
  - Implementation of the Next Steps in Signaling Protocols
  - Tested within testbeds and across the Internet

- Keep as much as possible of the existing implementation unmodified
  - Allow future versions of the implementation to be re-integrated without major adaptations

Institute of Telematics
Department of Computer Science

Comparison Study

- Comparison of NSIS-ka implementation with OMNeT++ simulation environment

	NSIS-ka	OMNeT++
Active entity	Thread	SimpleModule
Processing Mode	Parallel	Sequential/non-preemptive
Scheduling	Indirectly via thread conditions and synchronization	Directly on message arrival
Event signaling	Condition variables	Messages

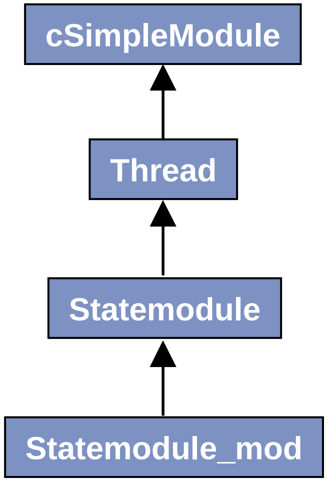
Institute of Telematics
Department of Computer Science

Design Decisions I

- Modeling of cSimpleModules
  - Model POSIX threads as cSimpleModules?
    - Pro – Logical coherence regarding messaging related entities
    - Con – No performance gain, much higher memory usage
  - Model entire NSIS module as cSimpleModule
  - `handleMessage()` realizes specific module’s logic
  - Thread synchronization realized by means of Boolean variables
    - No additional messages necessary
- Message handling
  - Insert messages for *FastQueue* directly into *Future Event Set*
  - Separate functions for message arrival and timeout handling in NSIS-ka implementation
    - `handleInternalMessage()` and `handleTimeout()`
    - Allows `handleMessage()` to call `handleInternalMessage()` directly upon message arrival

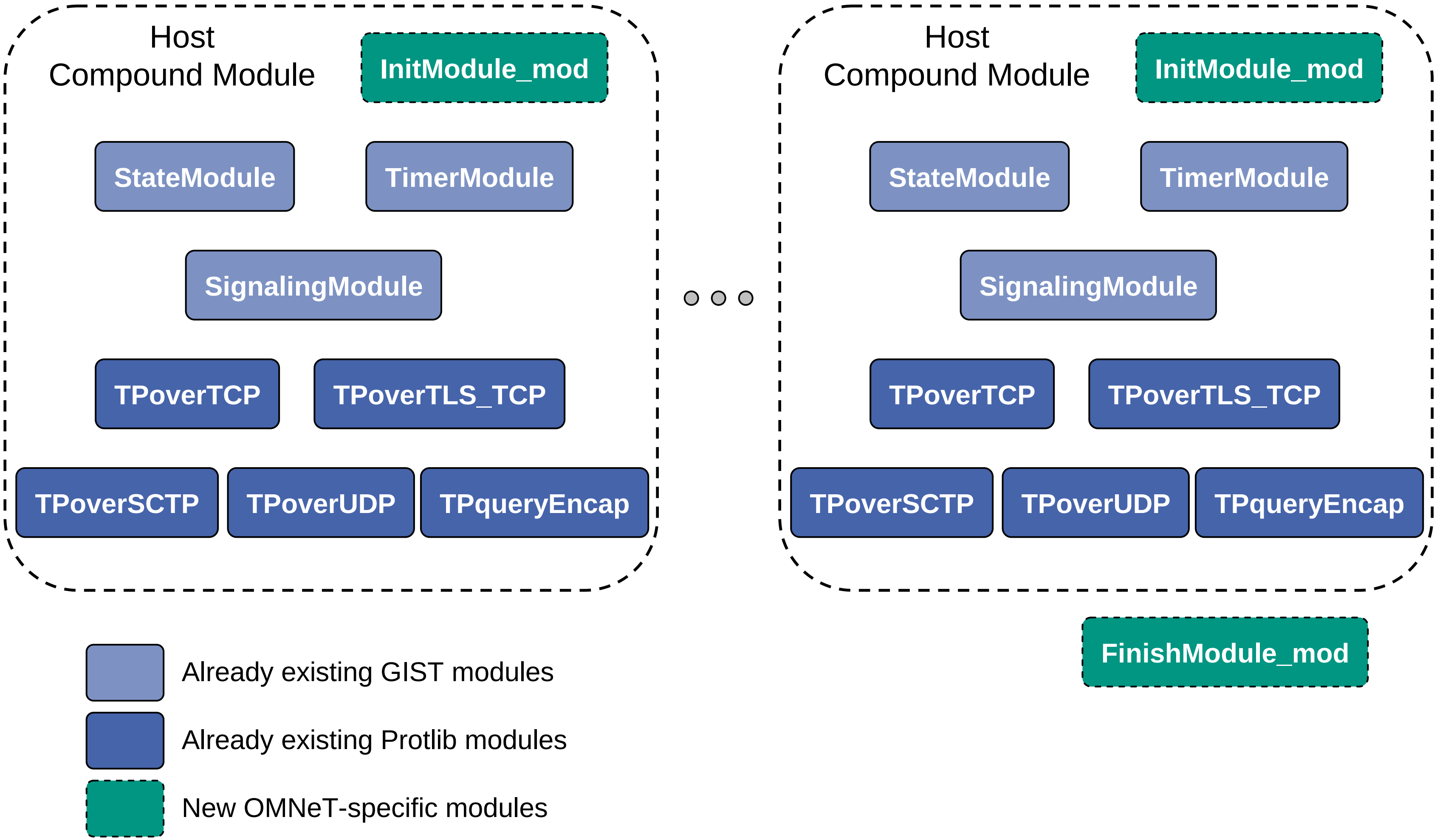
Design Decisions II

- Class hierarchy and module initialization
  - All modules must be subclassed by cSimpleModules
    - Empty constructor mandatory
      → initialization data cannot be passed to constructor
    - Introduce …module_mod class with empty constructor for each NSIS module
  - Data is not accessible via cSimpleModule constructor
    - Only within `initialize()` method
    - Allocate dedicated memory to superclass
      - Initialization within subclass by `initialize()` function
- Simulation of multiple hosts
  - Encapsulate modules of a host in CompoundModule
    - Each host must be clearly identifiable by unique NsisId
  - Introduce ModuleManager for initialization of each CompoundModule
    - Use dedicated init_module at beginning of each CompoundModule (i.e. host)



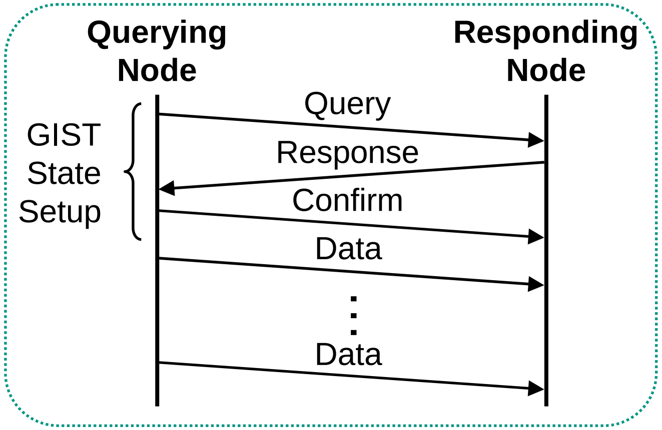
Institute of Telematics
Department of Computer Science

Necessary Module Extensions



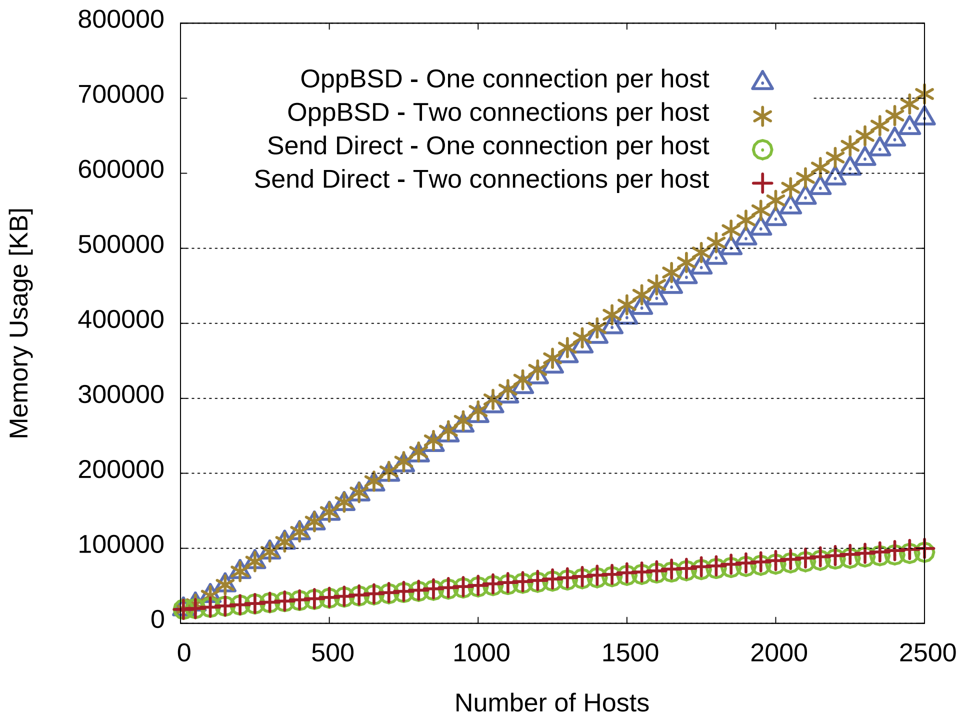
Evaluations

- Evaluation of Integration into OMNeT++
  - Not an evaluation of the GIST protocol implementation
- Evaluated by two different communication models
  - Abstract point-to-point communication model ("Send_Direct")
  - Real underlying FreeBSD TCP/IP Stack ("OppBSD")
    - Promises for more realistic simulation results
    - Allows analysis and validation of simulation results by means of tcpdump packet captures
- One or two connections per host
  - GIST state setup
  - 10 consecutive Data messages
  - Tested for a varying number of hosts



Evaluation Results – Memory Consumption

- Measurements for *Data Resident Set Size* (DRS)
  - Data segment of running application
- Linear growth rate
  - Constant memory consumption per host



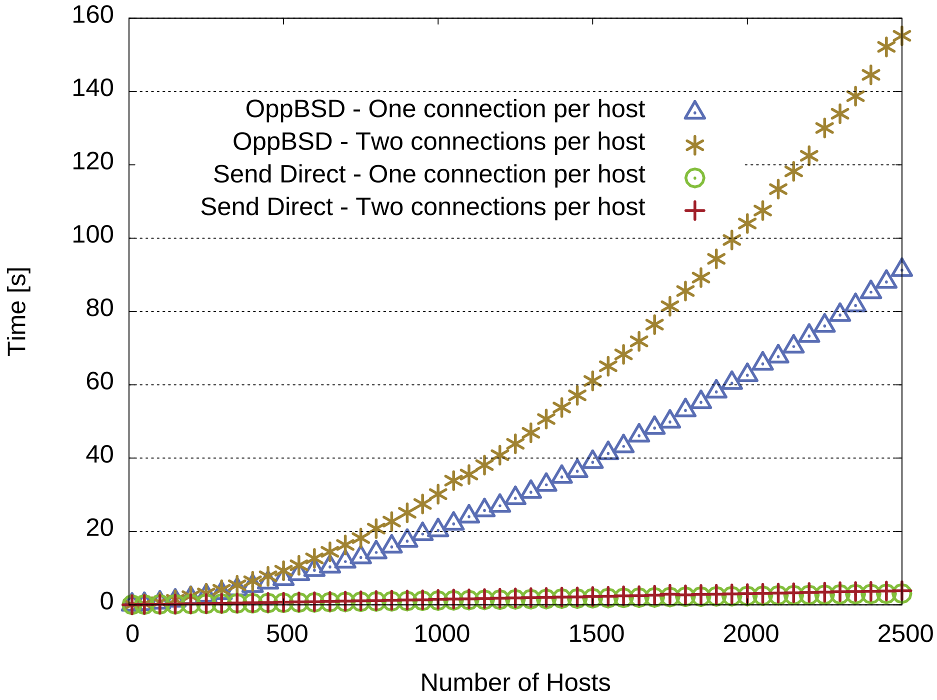
	Using OppBSD		Using Send_Direct	
# Hosts	One Conn.	Two Conn.	One Conn.	Two Conn.
10	18,825	18,961	18,634	18,638
100	36,653	36,801	21,366	21,634
1,000	276,485	283,077	48,702	50,646
10,000	751,853	758,469	324,194	345,994

- OppBSD causes much higher memory footprint due to allocated socket memory buffers

Institute of Telematics
Department of Computer Science

Evaluation Results – Runtime Performance

- Simulation of 130 seconds of protocol interaction



# Hosts	Using OppBSD One Conn.	Using OppBSD Two Conn.	Using Send_Direct One Conn.	Using Send_Direct Two Conn.
10	0.03	0.03	0.01	0.01
100	0.57	0.70	0.10	0.12
1,000	20.45	30.16	1.16	1.48
3,000	125.68	221.64	3.72	4.60

- 130 seconds simulated in *real-time* by simulating 3,500 hosts using OppBSD

Institute of Telematics
Department of Computer Science

Conclusion and Outlook

- Integration of existing implementation into simulation framework
  - Implementation already tested with real hardware
  - Simulation environment allows for greater flexibility and large-scale evaluations
- Use of OppBSD's TCP/IP stack promises realistic simulations
  - Possibility of obtaining tcpdump pcap files especially advantageous for offline analysis
- Future versions of the protocol implementation are directly integrated

Ongoing work

- Integration of NSLP implementations into OMNeT++
  - NAT/FW NSLP
  - QoS NSLP
- Use OMNeT++ topology generator ReaSE for large-scale protocol evaluations

Thank you for your attention

Questions?

Institute of Telematics
Department of Computer Science