Integration of a GIST implementation into OMNeT++

Roland Bless and Martin Röhricht
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
Comparison Study

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

<table>
<thead>
<tr>
<th></th>
<th>NSIS-ka</th>
<th>OMNeT++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active entity</td>
<td>Thread</td>
<td>SimpleModule</td>
</tr>
<tr>
<td>Processing Mode</td>
<td>Parallel</td>
<td>Sequential/non-preemptive</td>
</tr>
<tr>
<td>Scheduling</td>
<td>Indirectly via thread conditions</td>
<td>Directly on message arrival</td>
</tr>
<tr>
<td></td>
<td>and synchronization</td>
<td></td>
</tr>
<tr>
<td>Event signaling</td>
<td>Condition variables</td>
<td>Messages</td>
</tr>
</tbody>
</table>
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
    - handleMessage() 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)
Necessary Module Extensions

Host Compound Module

InitModule_mod

StateModule

TimerModule

SignalingModule

TPoverTCP

TPoverTLS_TCP

TPoverSCTP

TPoverUDP

TPqueryEncap

FinishModule_mod

Already existing GIST modules

Already existing Protlib modules

New OMNeT-specific modules
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

![Diagram of GIST state setup and communication between querying and responding node]
Evaluation Results – Memory Consumption

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

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

- OppBSD causes much higher memory footprint due to allocated socket memory buffers
Evaluation Results – Runtime Performance

Simulation of 130 seconds of protocol interaction

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

<table>
<thead>
<tr>
<th># Hosts</th>
<th>Using OppBSD</th>
<th>Using Send_Direct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>One Conn.</td>
<td>Two Conn.</td>
</tr>
<tr>
<td>10</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>100</td>
<td>0.57</td>
<td>0.70</td>
</tr>
<tr>
<td>1,000</td>
<td>20.45</td>
<td>30.16</td>
</tr>
<tr>
<td>3,000</td>
<td>125.68</td>
<td>221.64</td>
</tr>
</tbody>
</table>
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?