μIP Support for the Network Simulation Cradle

Michael Kirsche and Roman Kremmer
Computer Networks Communication Systems Group
Brandenburg University of Technology, Germany
### Excerpt of Available TCP/IP Stacks

<table>
<thead>
<tr>
<th>micro IP (μIP)</th>
<th>vs. lightweight IP (lwIP)</th>
<th>vs. FreeBSD IP-Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>○ 8-bit and 16-bit μC</td>
<td>○ Embedded hardware</td>
<td>○ 32-/64-bit systems</td>
</tr>
<tr>
<td>○ ~4KB RAM / ~10KB ROM</td>
<td>○ <del>20KB RAM/</del>/40KB ROM</td>
<td>○ KB → MBs RAM / ROM</td>
</tr>
<tr>
<td>○ Compliant with TCP, UDP and IP RFCs</td>
<td>+ Full-scale stack with DNS, PPP, ARP, DHCP, …</td>
<td>+ Full-scale stack with DNS, PPP, ARP, DHCP, …</td>
</tr>
<tr>
<td>+ Standalone version as well as Contiki integration</td>
<td>+ Standalone as well as OS support (multiple systems)</td>
<td>○ Embedded in FreeBSD</td>
</tr>
<tr>
<td>+ IPv6-ready (μIPv6)</td>
<td>– Experimental IPv6 support</td>
<td>+ Full IPv6 support</td>
</tr>
<tr>
<td>– Uses only 1 packet buffer, → throughput problems → AppLayer retransmission</td>
<td>+ High performance in almost all use cases</td>
<td>+ Highest performance in all use cases</td>
</tr>
<tr>
<td>– Standalone version does not support socket API</td>
<td>+ Socket as well as raw / native API for performance</td>
<td>– Requires Linux OS, no standalone support</td>
</tr>
</tbody>
</table>
# Excerpt of Available TCP/IP Stacks

<table>
<thead>
<tr>
<th>micro IP (μIP)</th>
<th>vs.</th>
<th>lightweight IP (lwIP)</th>
<th>vs.</th>
<th>FreeBSD IP-Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8-bit and 16-bit μC</strong></td>
<td></td>
<td><strong>Embedded hardware</strong></td>
<td></td>
<td><strong>32-/64-bit systems</strong></td>
</tr>
<tr>
<td><strong>~4KB RAM / ~10KB ROM</strong></td>
<td></td>
<td><strong>~20KB RAM/~40KB ROM</strong></td>
<td></td>
<td><strong>KB → MBs RAM / ROM</strong></td>
</tr>
<tr>
<td><strong>Compliant with TCP, UDP and IP RFCs</strong></td>
<td></td>
<td><strong>Full-scale stack with DNS, PPP, ARP, DHCP, …</strong></td>
<td></td>
<td><strong>Full-scale stack with DNS, PPP, ARP, DHCP, …</strong></td>
</tr>
<tr>
<td><strong>Standalone version as well as OS support (multiple systems)</strong></td>
<td></td>
<td><strong>Standalone as well as OS support (multiple systems)</strong></td>
<td></td>
<td><strong>Embedded in FreeBSD</strong></td>
</tr>
<tr>
<td><strong>IPv6-ready (μIPv6)</strong></td>
<td></td>
<td><strong>Experimenta IPv6 support</strong></td>
<td></td>
<td><strong>Full IPv6 support</strong></td>
</tr>
<tr>
<td><strong>Uses only 1 packet buffer, throughput problems</strong></td>
<td></td>
<td><strong>High performance in almost all use cases</strong></td>
<td></td>
<td><strong>High performance in all use cases</strong></td>
</tr>
<tr>
<td><strong>Standalone version does not support socket API</strong></td>
<td></td>
<td><strong>Socket as well as raw / native API for performance</strong></td>
<td></td>
<td><strong>Requires Linux OS, no standalone support</strong></td>
</tr>
</tbody>
</table>
Why microIP?

Sensor Node’s Stack

- CoAP
- MQTT
- uXMPP
- ... (Application Layer)
- UDP
- TCP (Transport Layer)
- uIPv6
- ICMPv6
- ND (Network Layer)
- MAC & PHY (e.g., IEEE 802.15.4-2006)

Gateway’s Stack

- uIPv6
- ICMPv6
- Routing
- ND
- 6LoWPAN (Network Layer)
- MAC & PHY (e.g., IEEE 802.15.4)

End-System’s Stack

- CoAP
- MQTT
- XMPP
- ... (Application Layer)
- UDP
- TCP (Transport Layer)
- Routing
- ND
- IPv6
- ICMPv6 (Network Layer)
- MAC & PHY (e.g., IEEE 802.3 / 802.11)

Part of a cyber physical system
Why simulate μIP in OMNeT++?

microIP is usually tested via:

1. Live experimentation on real systems
   - Deployments hard to control, low repeatability, costs, …

2. Testbeds
   - Low scalability, set-up inflexible, limited control of external factors, …

3. Cooja (Contiki OS simulator)
   - Cycle accurate emulation and possible interconnection to real systems
   - Limited simulation and comparison of/with systems/models outside the Contik world

Tackle these issues (+ more) through generic OMNeT++ simulation
What is the Network Simulation Cradle?

- Developed by Sam Jansen
- Basic idea: Integrate kernel-space implementations of real world network stacks into ns-2 / OMNeT++ \textit{(instead of failure prone / abstract modeling)}
- Basic approach:
  - Parse the C-code,
  - Substitute global variables through arrays of per-node-instance variables,
  - Recompile as a shared library,
  - Map interfaces to ns-2 / OMNeT++ through glue code.
- Works without manual code changes in contrast to “plain” porting of stacks
  - E.g.: Bless and Doll “\textit{Integration of the FreeBSD TCP/IP-Stack into the Discrete Event Simulator OMNet++}”
How to integrate μIP into the NSC?

- Process differs a bit for different microIP versions (w/o API)
  1. Integrate μIP source code into NSC build process
  2. Implement stubs for references to unused system functions
  3. Adjust globalizer parser for μIP
  4. Create new netstack drivers for Contiki (to redirect calls to NSC)
  5. Create config files and integrate into an OMNeT++ simulation
How to use μIP in OMNeT++?

- Prerequisite:
  - 32-bit Linux → NSC requirement
  - Source code from Github

- Steps:
  - Copy our code into extracted NSC
  - Compile shared μIP library (libuip.so / libuipv6.so)
  - Adjust LD_LIBRARY_PATH
  - Enable NSC / recompile simulation
  - Setup omnetpp.ini
In Conclusion

• μIP support in OMNeT++:
  • Provided via the NSC
  • Currently support for IPv4
  • Packet exchange between different stacks possible

  ➢ Another stop along the road of IoT simulations with OMNeT++

• Further actions:
  • Full IPv6 integration (NSC officially has IPv6 support, function calls yet always go to v4)
  • Combine with IEEE 802.15.4, 6LoWPAN and applayer protocol

  ➢ Possible integration of other stacks in the future (e.g., RiotOS)

Thank you for your attention!  Questions?