



1



2nd OMNeT++ Community Summit 2015

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

VS.



micro IP (µIP)

- $\odot~$ 8-bit and 16-bit μC
- ~4KB RAM / ~10KB ROM
- Compliant with TCP, UDP and IP RFCs
- + Standalone version as well as Contiki integration
- + IPv6-ready (µIPv6)
- Uses only 1 packet buffer,
 → throughput problems
 → AppLayer retransmission
- Standalone version does not support socket API

lightweight IP (IwIP) vs.

- o Embedded hardware
- ~20KB RAM/~40KB ROM
- + Full-scale stack with DNS, PPP, ARP, DHCP, ...
- + Standalone as well as OS support (multiple systems)
- Experimental IPv6 support
- + High performance in almost all use cases
- + Socket as well as raw / native API for performance

FreeBSD IP-Stack

- o 32-/64-bit systems
- \circ KB → MBs RAM / ROM
- + Full-scale stack with DNS, PPP, ARP, DHCP, ...
- o Embedded in FreeBSD
- + Full IPv6 support
- + Highest performance in all use cases
- Requires Linux OS, no standalone support

Excerpt of Available TCP/IP Stacks



micro IP (µIP)

- $\odot~$ 8-bit and 16-bit μC
- ~4KB RAM / ~10KB ROM
- Compliant with TCP, UDP and IP RFCs
- + Standalors O: O as well
- + IPv6-ready
- Use h → packet buffer,
 → throughput problems
 → AppLayer retransmission
- Standalone version does not support socket API

vs. lightweight IP (IwIP) vs.

- o Embedded hardware
- ~20KB RAM/~40KB ROM
- + Full-scale stack with CIS, PPP, ARP Directed States
- support (multiple systems)

+ High perí racen Cases

+ Socket as well as raw / native API for performance

FreeBSD IP-Stack

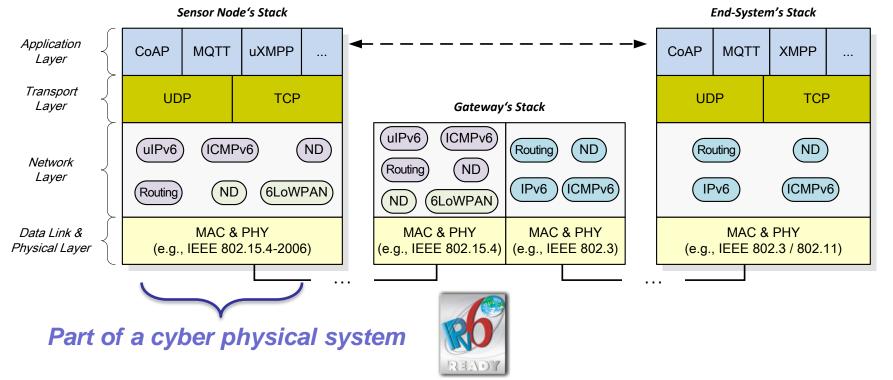
o 32-/64-bit systems

KB → MBs RAM / RCM
 + Full-scale of closed on the second sec

- Emberded in FreeBSD
- Requires Linux OS, no standalone support

Why microlP ?





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

Stackle these issues (+ more) through generic OMNeT++ simulation

What is the Network Simulation Cradle ?

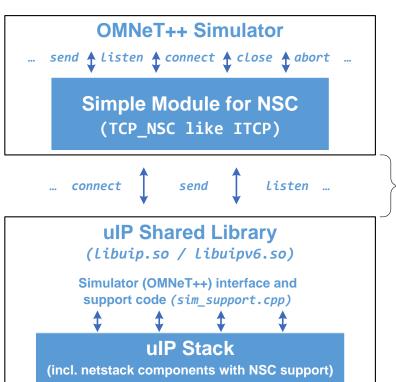


- Developed by Sam Jansen
- Basic idea: Integrate kernel-space implementations of real world network stacks into ns-2 / OMNeT++ (instead of failure prone / abstract modeling)
- Basic approach:
 - Parse the C-code,
 - Substitute global variables through arrays of per-node-instance variables,
 - Recompile 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 "Integration of the FreeBSD TCP/IP-Stack into the Discrete Event Simulator OMNet++"

How to integrate μ IP into the NSC ?



- Process differs a bit for different microIP versions (w/o API)
- 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 μ IP in OMNeT++ ?



- Prerequisite:
 - 32-bit Linux \rightarrow 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

<pre>Properties for inst-michs Project Features Project</pre>	COLUMN TE
<pre> Resource Builders buildime. Feature definitions come from the project's ".oppfeatures' file. builders builder</pre>	×
<pre>> Resource Builders build time. Feature definitions come from the project's "oppfeatures" file. > C/C++ Build > C/C++ Build > C/C++ General > cpcheclipse Git Unux Tools Path • OMNET++ Documentation Genera TCP (NRC) Make 1 [Config default_tCp] NED 2 description = "inet_TCP <> default_tCP" * CP (NRC) * description = "inet_TCP <> default_tCP" * default TCP implementation 7 cp (NRC) * default TCP implementation * *.tcpType = "tCP" * default TCP implementation 1 cm fig inetinet_TCP * *.tcpType = "tCP" * *.tcpType = "tCP"</pre>	I
<pre>> cppcheclipse Git Linux Tools Path</pre>	:en
Gr Linux Tools Path ↓ ONNeT++ Documentation Genera Make 1 [Config default_top] NED 2 description = "default_TCP" ↓ [Config inet_inet] Projet # 5 description = "inet_TCP <> default_TCP" ↓ [Config inet_inet] Projet # 5 description = "inet_TCP" & accurrent for the fault of the fa	All
<pre> a ONNeT++ Documentation Genera Documentation = "default_TCP" Project = \$ description = "inet_TCP" Run/Deb 6 # default_TCP implementation T **.tcpTpe = "TCP" if # default_TCP implementation 12 **.tcpTpe = "TCP" if # default_TCP implementation 12 **.tcpTpe = "TCP" if # default_TCP implementation 12 **.tcpTpe = "TCP" if # default_TCP implementation description = "inet_TCP de = "object" # bytecount object bytestream if [Config inet_inet_stream] description = "inet_TCP -> inet_TCP" if # default_TCP constant if # default_TCP constant if # default_TCP implementation if # default_TCP implementatio</pre>	
Documentation Genera TCP (NSC) Mak 1 [Coffig default_CP] NED 2 description = "default_TCP" Projet 4 [Coffig inetinet] Projet 5 description = "inet_TCP <> inet_TCP" Run/De 6 # default TCP implementation 7 **.tcpType = "TCP" 9 [Config inetinet_msg] 1 description = "inet_TCP <> inet_TCP" 1 # default TCP implementation 1 **.tcpType = "TCP" 1 # default TCP implementAtion 1 description = "inet_TCP <> inet_TCP" 1 # default TCP implementAtion 1 description = "inet_TCP <> inet_TCP" 1 # default TCP implementAtion 1 **.tcpType = "TCP"	T All
<pre>Makt 1 [Config default_tcp] NED 2 description = "default_TCP" Project 8 { Generation = "inet_TCP <> inet_TCP" Run/Det 6 # default_TCP implementation 7 **.tcpType = "tCP" 9 [Config inet_inet_TCP" 9 [Config inet_inet_TCP" 1 # default_TCP implementation 12 **.tcpType = "tCP" 13 **.tcpType = "tCP" 14 # default_TCP implementation 12 **.tcpType = "tCP" 13 **.tcpType = "tCP" 14 default_TCP implementation 15 [Config inet_inet_stream] 16 description = "inet_TCP <> inet_TCP" 17 # default_TCP implementation 18 **.tcpType = "tCP"</pre>	
<pre>NED 2 description = "default_TCP <> default_TCP" Projet 4 Projet 5 description = "int TCP <> inteTCP" Run/Deb 6 # default TCP implementation 7 **.tcpType = "TCP" 6 [Config inet_inet_msg] 10 description = "inteTCP <> inteTCP" 11 # default TCP implementation 12 **.tcpType = "TCP" 13 **.tcpType = "TCP" 14 [Config inet_inet_stream] 16 description = "inteTCP <> inteTCP" 17 # default TCP implementation 18 **.tcpType = "TCP" </pre>	
<pre> [Proje 4 [Config inet_inet] Projet 5 description = "inet_TCP" Run/Deb 6 # default TCP implementation 7 **.tcpType = "TCP" 8 9 [Config inet_inet_meg] 10 description = "inet_TCP> inet_TCP" 11 # default TCP implementation 12 **.tcpType = "TCP" 13 **.tcpType = "TCP" 14 15 [Config inet_inet_stream] 16 description = "inet_TCP <> inet_TCP" 17 # default TCP implementation 18 **.tcpType = "TCP"</pre>	
<pre>Project R 5 description = "inet_TCP <> inet_TCP" Run/De 6 # default TCP implementation 7 **.tcpType = "TCP" 8 9 [Config inetinet_msg] 10 description = "inet_TCP <> inet_TCP" 11 # default TCP implementation 12 **.tcpType = "TCP" 13 **.tcpType = "TCP" 14 **.tcpType = "TCP" 15 **.tcpType = "TCP" 16 description = "inet_TCP <> inet_TCP" 17 # default TCP implementation 18 **.tcpType = "TCP" 17 # default TCP implementation 18 **.tcpType = "TCP"</pre>	
<pre>7 **.tcpType = "TCP" 8 9 [Config inet_inet_msg] 10 description = "inet_TCP" 11 # defoult TCP implementation 12 **.tcpType = "TCP" 13 **.tcpType = "TCP" 14 15 [Config inet_inet_stream] 16 description = "inet_TCP <> inet_TCP" 17 # defoult TCP implementation 18 **.tcpType = "TCP"</pre>	
<pre>8 9 [Config inet_inet_msg] 10 description = "inet ICP <> inet_TCP" 11 # default TCP implementation 12 **.tcpType = "TCP" 13 **.tcpType[*].dataTransferMode = "object" # bytecount / object / bytestream 14 15 [Config inet_inet_stream] 16 description = "inet_TCP <> inet_TCP" 17 # default TCP implementation 18 **.tcpType = "ICP"</pre>	
<pre>10 description = "inet_TCP <> Snet_TCP" 11 # default TCP implementation 12 **.tcpType = "TCP" 13 **.tcpApp[*].dataTransferMode = "object" # bytecount object bytestream 14 15 [Config inet_inet_stream] 16 description = "inet_TCP <> inet_TCP" 17 # default TCP implementation 18 **.tcpType = "TCP"</pre>	
<pre>11 # default TCP implementation 12 **.tcpType = "TCP" 13 **.tcpType[*].ddsTransferWode = "object" # bytecount object bytestream 14 15 [Config inet_inet_stream] 16 description = "inet_TCP <> inet_TCP" 17 # default TCP implementation 18 **.tcpType = "TCP"</pre>	
<pre>13 **.tcpApp[*].dataTransferMode = "object" # bytecount object bytestream 14 15 [Config inet_inet_stream] 16 description = "inet_TCP </pre> 17 # default TCP implementation 18 **.tcpType = "TCP"	
<pre>14 15 [Config inet_inet_stream] 16 description = "inet_TCP <> inet_TCP" 17 # default TCP implementation 18 **.tcpType = "TCP"</pre>	
16 description = "inst_TCP <> inet_TCP" 17 # default TCP implementation 18 **.tcpType = "TCP"	
17 # default TCP implementation 18 **.tcpType = "TCP"	
18 **.tcpType = "TCP"	
19 "*.tcpAppl*1.datairansterMode = "bytestream" # bytecount object bytestream	
20	
21 [Config NSClinux_NSClinux]	
22 description = "NSC_TCP (Linux) <> NSC_TCP (Linux)" 23 # setting TCP stack implementation	
23 * secting the state implementation	
25	
<pre>26 [Config NSClinux_NSClinux_stream] 27 description = "NSC TCP (Linux) <> NSC TCP (Linux)"</pre>	
27 description = ms_lter (Linux) <> ms_lter (Linux) 28 # setting TCP stack implementation	

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?