Simulation with Real World Network Stacks

Sam Jansen and Anthony McGregor
WAND Network Research Group
University of Waikato
TCP models and network simulation

• What?
  • We use direct execution of real world TCP code for TCP models

• Why model and simulate networks with TCP?
  • Allows experimentation of network protocols in many scenarios, including ones that cannot currently be built
  • TCP models especially used a lot by Internet researchers
  • TCP forms the basic transport protocol used for a vast proportion of the data moved on the Internet
Existing TCP models

• ns-2 most popular network simulator for TCP simulation
  • Even its models are lacking
  • Doesn't include some features (eg. receivers advertised window/flow control)
  • Some of its limited TCP models are well validated, others not so well
  • Compared to other network simulators its TCP models are considered very good and featureful – yet does not capture network stacks running on real computers in some important ways

• Real world TCP/IP stacks are a moving target and vary amongst themselves
  • ns-2's models form a fairly good abstraction, but not always good enough for a researcher
Case study – uniform loss

• During 5% uniform random loss, ns-2 simulated results compared with measured results from a testbed network:

<table>
<thead>
<tr>
<th>TCP Implementation</th>
<th>Goodput (kb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns-2: Sack1/Sack1-DelAck</td>
<td>88.9</td>
</tr>
<tr>
<td>Linux 2.4.27</td>
<td>208.6</td>
</tr>
<tr>
<td>ns-2: Newreno/DelAck</td>
<td>96.9</td>
</tr>
<tr>
<td>Linux 2.4.27 no SACK</td>
<td>193.7</td>
</tr>
<tr>
<td>FreeBSD 5.2.1</td>
<td>162.8</td>
</tr>
</tbody>
</table>
Things could be better

• We'd like to be able to use all those network stacks we measured on our testbed in simulation

• Others have included a single network stack in simulators before, but there have been limitations:
  • have not created a maintainable way of keeping stacks up to date
  • have not made it easy to add new network stacks
  • only integrate with one simulator
  • little or no validation
Related work

• Integration of the FreeBSD TCP/IP-Stack into the Discrete Event Simulator OMNet++. Roland Bless and Mark Doll (University of Karlsruhe). WSC'04.
  
  • Being worked on as we integrated FreeBSD into ns-2!

  
  • Requires patched kernel on simulation machines, requires one machine per different stack simulated
The Network Simulation Cradle

• The Network Simulation Cradle allows simulation with multiple different network stacks on the same machine

• It is efficient and scalable

• It provides a tool to ease the process of supporting multiple instances of network stacks

• Seamlessly integrated with ns-2 and provides a general interface to allow integration with other packet based network simulators

• Currently supports the following TCP/IP stacks:
  • FreeBSB, OpenBSD, Linux (2.4.x and 2.6.x) and lwIP
How?

• Extract the network stack from its host system and build a “cradle” around it

Diagram:

- Network Stack
- Operating system kernel
- Kernel space
- Cradle
- User space
Cradle creation

• Creating a cradle and incorporating a new stack is a process that requires some technical knowledge and can be time consuming
  • Host system support code
  • Stub functions
  • NSC specific functions
• After the initial creation of NSC using the FreeBSD stack, subsequent additions have taken a week or two
• Updating stacks is easy, we have quickly updated between several versions of Linux and FreeBSD without problem
Simulator / cradle communication

- Network simulator (ns-2)
- NSC TCP model
- Network Stack
- Cradle (shared lib)
Simulator / cradle communication

- The cradle and network stack is contained in a shared library (dynamic link library)

- The network simulator has an agent which routes important simulation events through a standard C++ interface

- Each network stack will have some support or “cradling” code which implements this C++ interface

- Each network simulator will have some agent code which implements the simulator side of the C++ interface and integrates as a TCP model of that simulator

- But we need to support multiple instances of network stacks some way: can't have a separate shared library for each instance
Multiple instances

- The earlier diagram of creating a cradle showed how the cradle supported the code for a single network stack.
- This is true, but we wish to support many (thousands) of instances of each type of stack.
- The network stacks are designed to be run on but one computer, there is no support for running many concurrently in the same program space.
- Earlier examples of similar work have modified the source code by hand to "virtualise" it – to allow multiple instances to run at the same time without affecting results.
Global variables and a parser

- The problem lies in global variables in the source code. These variables need to be mapped through some indirection table.

- Modifying code by hand not maintainable, requires a lot of time spent initially, and easy to make errors

- Instead: solve the problem programmatically

- Create a parser to understand the input C source code and modify global variables and their references

- Our global parser, or globaliser, understands most of C99 and gcc extensions to C and is able to parse the network stacks and related code of the FreeBSD, OpenBSD and Linux kernels: a large amount of code!
Seamless ns-2 integration

- ns-2 is supported seamlessly: our network stacks have a similar interface to existing ns-2 TCP models

- Example simulation code: FTP over TCP

Original simulation code

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set tcp [new Agent/TCP]</td>
<td></td>
</tr>
<tr>
<td>$ns attach-agent $n0 $tcp</td>
<td></td>
</tr>
<tr>
<td>set sink [new Agent/TCPSink]</td>
<td></td>
</tr>
<tr>
<td>$ns attach-agent $n4 $sink</td>
<td></td>
</tr>
<tr>
<td>$ns connect $tcp $sink</td>
<td></td>
</tr>
<tr>
<td>$tcp set fid_ 1</td>
<td></td>
</tr>
<tr>
<td>$tcp set packetSize_ 552</td>
<td></td>
</tr>
<tr>
<td>set ftp [new Application/FTP]</td>
<td></td>
</tr>
<tr>
<td>$ftp attach-agent $tcp</td>
<td></td>
</tr>
</tbody>
</table>

Simulation code using NSC TCP models

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set tcp [new Agent/TCP/NSC/FreeBSD5]</td>
<td></td>
</tr>
<tr>
<td>$ns attach-agent $n0 $tcp</td>
<td></td>
</tr>
<tr>
<td>set sink [new Agent/TCP/NSC/Linux24]</td>
<td></td>
</tr>
<tr>
<td>$ns attach-agent $n4 $sink</td>
<td></td>
</tr>
<tr>
<td>$ns connect $tcp $sink</td>
<td></td>
</tr>
<tr>
<td>$tcp set fid_ 1</td>
<td></td>
</tr>
<tr>
<td>$tcp set packetSize_ 552</td>
<td></td>
</tr>
<tr>
<td>set ftp [new Application/FTP]</td>
<td></td>
</tr>
<tr>
<td>$ftp attach-agent $tcp</td>
<td></td>
</tr>
</tbody>
</table>
Limitations

• Require the source code for the network stack

• Performance?
  • Uses real packets, meaning entire packets are assembled and the data is copied about the simulator
  • Less abstraction, more complex code to run through, slower?

• Scalability?
  • Memory usage is a lot larger due to the real packets and other reasons: can't support as many TCP endpoints
Performance

• Slower than existing TCP models which are abstracted a lot more, but not as slow as we thought it would be

• The difference varies, in some situations the existing ns-2 TCP models are twice as fast, in others 4 times, in others less than twice

• In a recent set of simulations we found the difference in CPU time used to be less than twice for the vast majority of instances

• Using NSC stacks is slower, but in most cases the slowdown is probably not significant

• Because NSC stacks integrate seamlessly, they can be used as a basic validation test: only use them some of the time, use the existing TCP models the rest of the time
Scalability

• NSC stacks use significantly more RAM

• Real packets in memory means there is potential for a large amount of memory needed on one machine: limits simulation size

• We run simulations with hundreds of TCP endpoints on a daily basis and have tested with a few thousand with success in the past

• Again: more limited than the existing models, but probably not a problem for many researchers out there – lots of TCP research does not use all that many flows
Example results: random loss revisited

- NSC simulation results vs. testbed measured results

<table>
<thead>
<tr>
<th>TCP Implementation</th>
<th>Goodput (kb/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenBSD 3.5</td>
<td>113.2</td>
</tr>
<tr>
<td>NSC: OpenBSD 3.5</td>
<td>109.0</td>
</tr>
<tr>
<td>Linux 2.4.27</td>
<td>208.6</td>
</tr>
<tr>
<td>NSC: Linux 2.4.27</td>
<td>204.9</td>
</tr>
<tr>
<td>Linux 2.4.27, No SACK</td>
<td>193.7</td>
</tr>
<tr>
<td>NSC: Linux 2.4.27, No SACK</td>
<td>192.7</td>
</tr>
<tr>
<td>FreeBSD 5.2.1</td>
<td>162.8</td>
</tr>
<tr>
<td>NSC: FreeBSD 5.2.1</td>
<td>156.5</td>
</tr>
</tbody>
</table>
How different are they?

• Previous slide showed a difference between measured and simulated results of up to 4%

• Why?
  • Application models
  • Still other abstractions

• Can we do better?
  • Move more code into simulation

• But it is pretty good as it is...
How different: part 2

Time Sequence Graphs of Simulated and Real FreeBSD

Time Sequence Graphs of Simulated and Real Linux
Summary

• Accurate simulation using direct execution of TCP code from real stacks – the very code that is run on real machines

• Architecture to support:
  • many network stacks
  • ease of integration of new network stacks
  • ease of updating network stacks

• Some limitations in performance and scalability, but practically not too bad

• Simulator independent

• Currently uses ns-2 simulator and works alongside existing TCP models