Simulation Speedup of ns-3 using Checkpoint and Restore

Kyle Harrigan

kyle.harrigan@gtri.gatech.edu

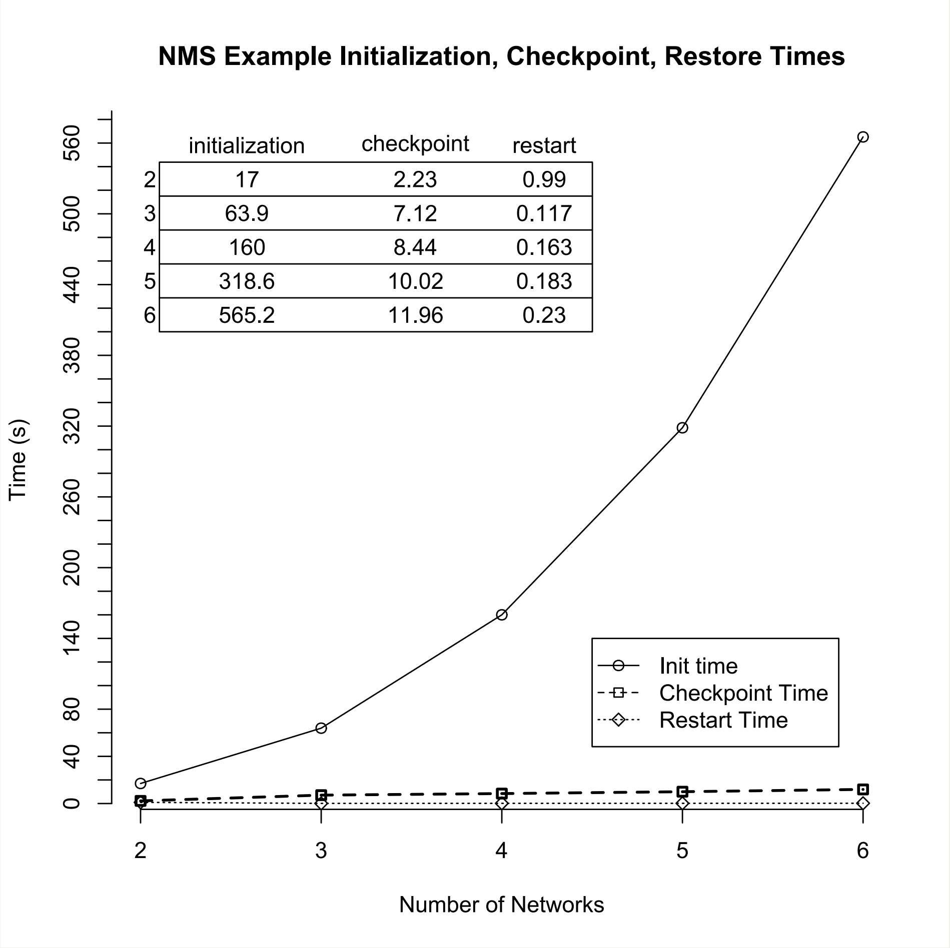
Overview

- The Problem
- The Concept
- Related Work
- Contributions
- Results
- Conclusions / Future Work

The Problem

A significant amount of computational resources can be wasted performing unnecessary and/or repetitive computations

Example: NMS Campus Network Routing Table Calculation

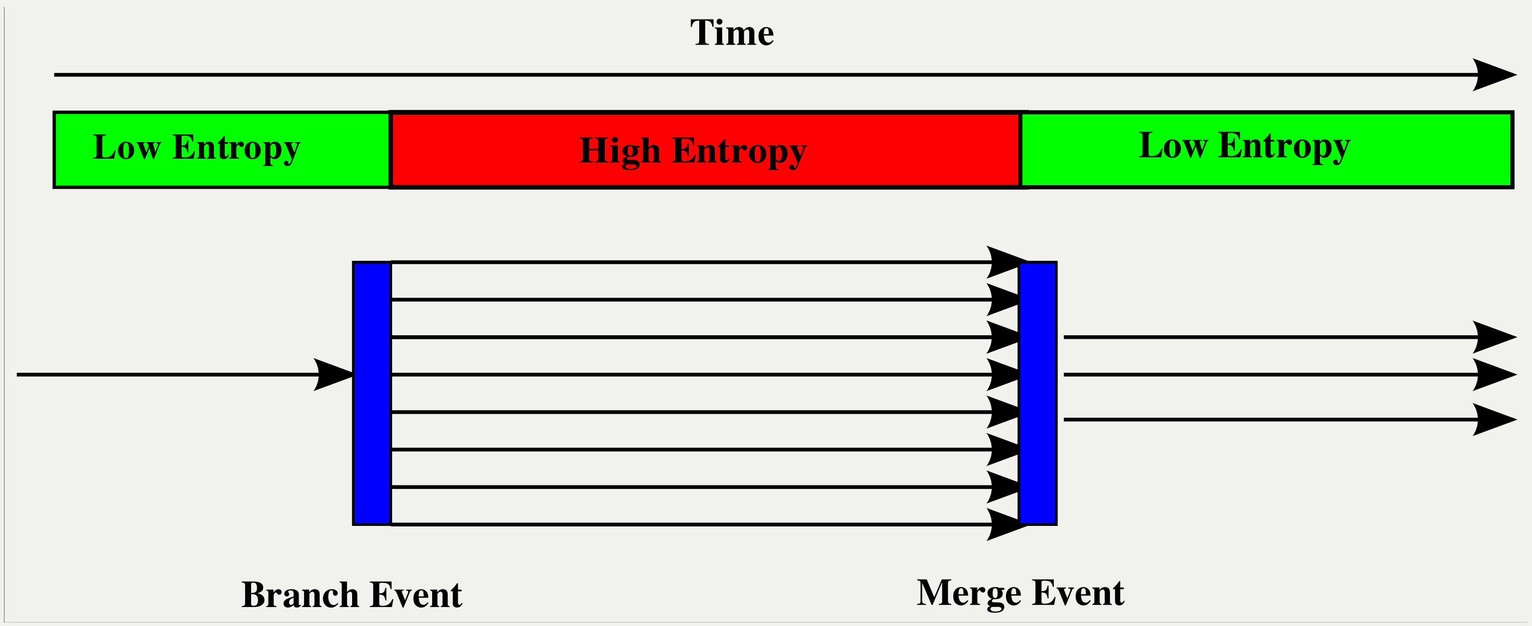


	initialization	checkpoint	restart
2	17	2.23	0.99
3	63.9	7.12	0.117
4	160	8.44	0.163
5	318.6	10.02	0.183
6	565.2	11.96	0.23

nix vector routing disabled, explicitly compute routing tables

Other examples

- In Monte Carlo analysis, early computations can eat up a significant amount of runtime, over and over again
- Some run segments may have little or no variability across repeated trials
- Rare events, importance sampling (Glasserman, et. al.)
- Maybe more like this:



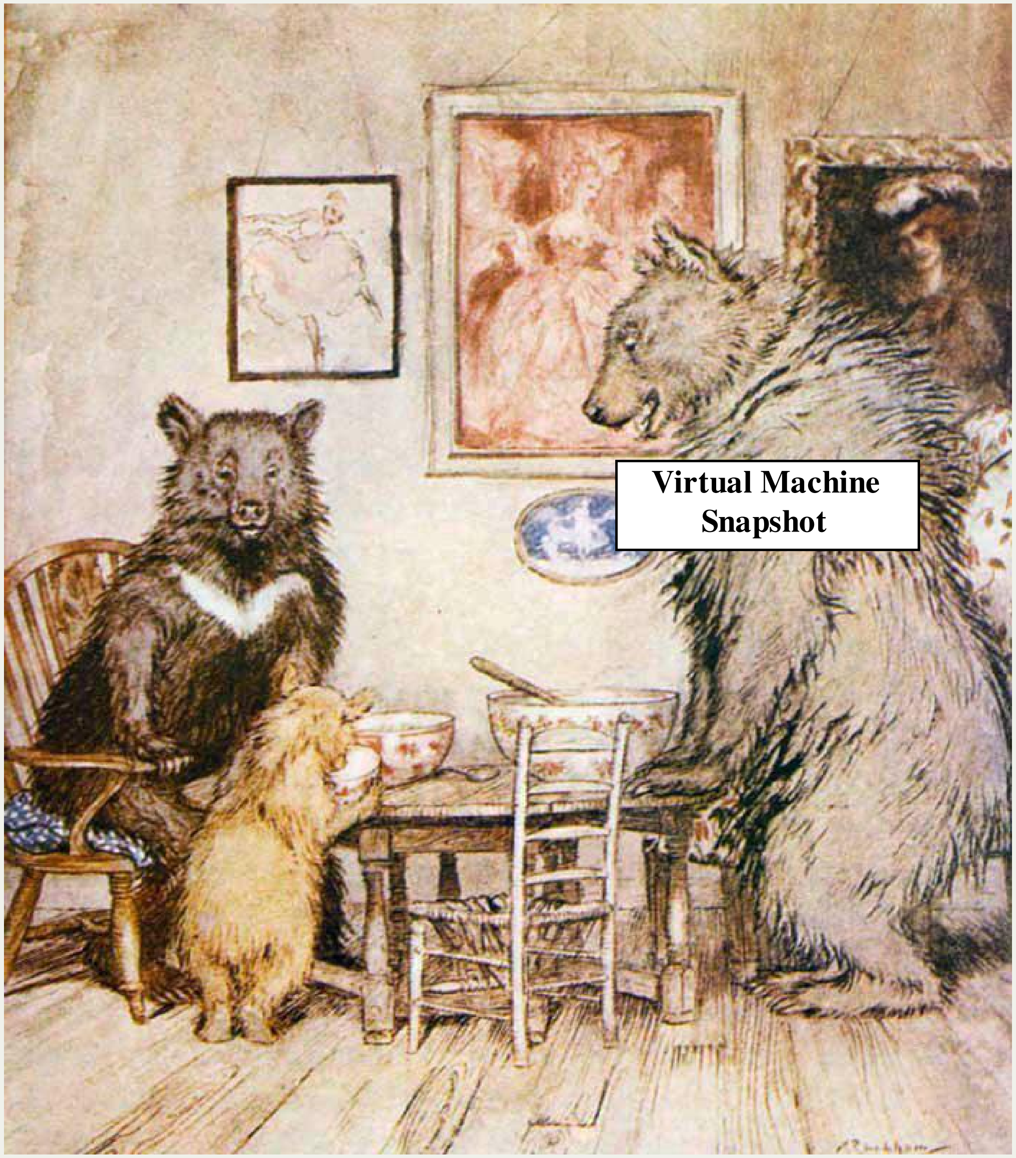
Concept

In PDES, we often attempt to improve simulation runtime through parallelization efforts:

- OpenMP/OpenMPI
- HPC clusters
- GPUs

Also useful to minimize time spent evaluating "uninteresting" segments by providing a generic ability to checkpoint previous runs and restart using different algorithms / parameters?

Candidate technologies for checkpointing



Process Checkpointing

- Use process checkpointing to save simulation state after costly computations or at decision points
- Good balance between heavy-handed VM checkpoints and custom application-dependent state saving
- Permit modification of parameters upon restart
  - (Not very useful or interesting without this!...)
- **Initially**: Use as tool to avoid repetitive computations
- **Eventually**: As a method for interactive analysis (Simulation Cloning), closed-loop algorithm optimization, etc.

Previous Work

Simulation Cloning

Simulation cloning is a novel method for interactively testing alternative scenarios in parallel simulations based on the concept of branching at decision points.

Maria Hybinette and Richard M. Fujimoto. 2001. Cloning parallel simulations. ACM Trans. Model. Comput. Simul. 11, 4 (October 2001), 378-407. DOI=10.1145/508366.508370 http://doi.acm.org/10.1145/508366.508370

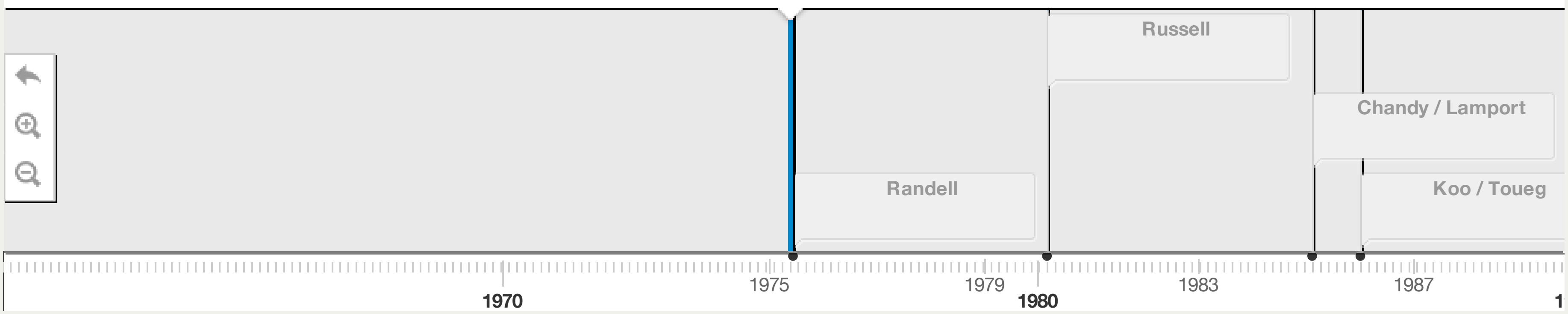
A History of Checkpointing

Checkpoint and restore utilities have a long history of use in fault-tolerant computing, dating **at least** as far back to work in (Chandy, 1985), (Koo, 1986) with many more developments since.

It was long used before this in database recovery.

Process checkpoint/restore is perhaps more recent due to the massive recent increase in storage capability

JUNE 1, 1975
Randell



Why DMTCP?

- Works entirely in userspace (no kernel modification)
- Supports Multithreaded, MPI, Distributed Applications
- Supports many popular scientific applications
  - MATLAB, Python, Perl, PHP, Emacs, GHCi, gnuplot, Lynx, Octave, Ruby...
- Handles fork, exec, ssh, mutex/semaphore, sockets, pipes, file descriptors, shared memory, etc.
- Minimal code modifications required (unless you choose to)

What makes this approach different?

- Focus on simplicity, generality, ease of use
- Assume minimal changes to underlying codebase as requirement
  - In some cases, little/no access to source code
  - In most cases, little money, time, or ability to modify
- Less focus on HLA, RTI, distributed simulation and more on speedup of parallel cluster or even desktop computations

Contributions

- Demonstrated successful checkpointing and restart of ns-3 in userspace using DMTCP libraries
- Demonstrated modification of parameters in restarted simulations using Namespace-based Access
- Documented checkpoint sizes and times for various ns-3 examples
- Compared checkpoint sizes and times for various compression schemes

Checkpointing "Module"

- Required steps / modifications?
  - install dmtcp (http://dmtcp.sourceforge.net/)
  - `checkpointer.h`
  - `checkpointer.cc`
  - Updates to wscript
- In other words...not much (I consider this good)

In ns-3 script:

```
#include <checkpointer.h>

Checkpointer cp;
cp.CheckpointAt(0.5); // schedule a checkpoint at 0.5 seconds
```

Checkpoint, Modify, Restore

Launch example in the DMTCP environment:

```
$ dmtcp_launch --modify-env build/scratch/first-cp
```

Prior to restart, make a `dmtcp_env.txt` in the ckpt folder:

```
NS3_ARGUMENTS="/NodeList/*/DeviceList/0/DataRate/10Kbps:/ChannelList/0/Delay/
```

Then, restart to run w/ modified parameters.

```
$ ./dmtcp_restart_script.sh --ckptdir .
```

Launch MPI job:

```
$ dmtcp_launch mpirun -np 2 ./waf --run simple-distributed
```

How

DMTCP places contents of dmtcp_env.txt into restarted environment

```
char *env = getenv(NS3_ARGUMENTS);
```

Parse this, set ns-3 configuration variables using Namespace-based Access (could easily use other methods)

```
// parse into key value pairs, then..
Config::Set(key, StringValue(value));
```

Results

Baseline run (Rate=5Mbps, Delay=2ms)

```
$ dmtcp_launch --modify-env build/scratch/first-cp

At time 2s client sent 1024 bytes to 10.1.1.02 port 9
At time 2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

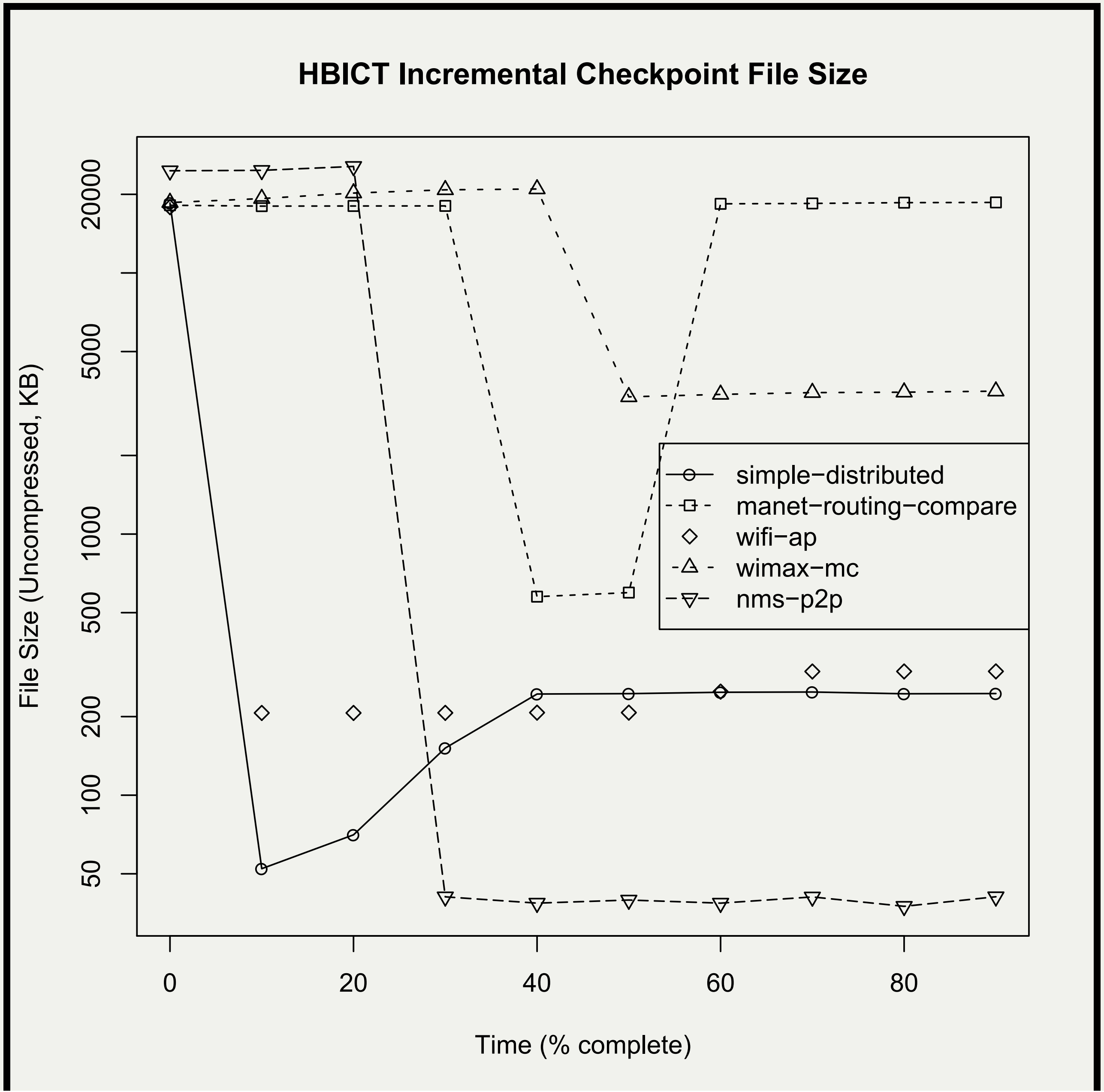
Modified upon restart using dmtcp_env.txt:

```
NS3_ARGUMENTS="/NodeList/*/DeviceList/0/DataRate/10Kbps:/ChannelList/0/Delay/

$ ./dmtcp restart script.sh --ckptdir .
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.8452s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.8452s server sent 1024 bytes to 10.1.1.1 port 49153
At time 3.6904s client received 1024 bytes from 10.1.1.2 port 9
```

Checkpoint Sizes

Example Name	MPI	Size	Size (Gzip)
first	No	56MB	17MB
simple-error-model	No	56MB	17MB
energy-model-example	No	56MB	17MB
radvd-two-prefix	No	56MB	17MB
hello-simulator	No	59MB	17MB
matrix-topology	No	58M	18MB
nms-p2p-nix-distributed	No	86MB	23MB
simple-distributed	Yes	198MB	65MB
nms-p2p-nix-distributed	Yes	240MB	100MB


HBICT Incremental Checkpoint File Size
File Size (Uncompressed, KB)
Time (% complete)
20000
5000
1000
500
200
100
50
0
20
40
60
80
simple−distributed
manet−routing−compare
wifi−ap
wimax−mc
nms−p2p

Checkpoint Times

Example Name	No Compression				Gzip				HBICT			
	min	max	mean	sd	min	max	mean	sd	min	max	mean	sd
wifi-ap	0.308	6.808	1.002	0.772	1.531	4.657	3.186	0.746	0.286	13.68	1.167	2.468
nms-p2p-nix-distributed	0.451	2.696	0.608	0.353	1.733	6.816	2.664	0.601	0.442	4.729	0.901	0.798
manet-routing-compare	0.304	3.459	0.681	0.456	1.222	25.163	4.052	4.730	0.291	7.627	0.734	1.204
wimax-multicast	0.304	2.702	0.687	0.351	1.343	26.770	5.384	6.119	0.305	8.095	0.912	1.238
simple-distributed (MPI)	0.436	6.608	1.244	1.760	1.624	2.415	2.109	0.148	0.536	4.541	0.911	1.015

Conclusions / Future Work

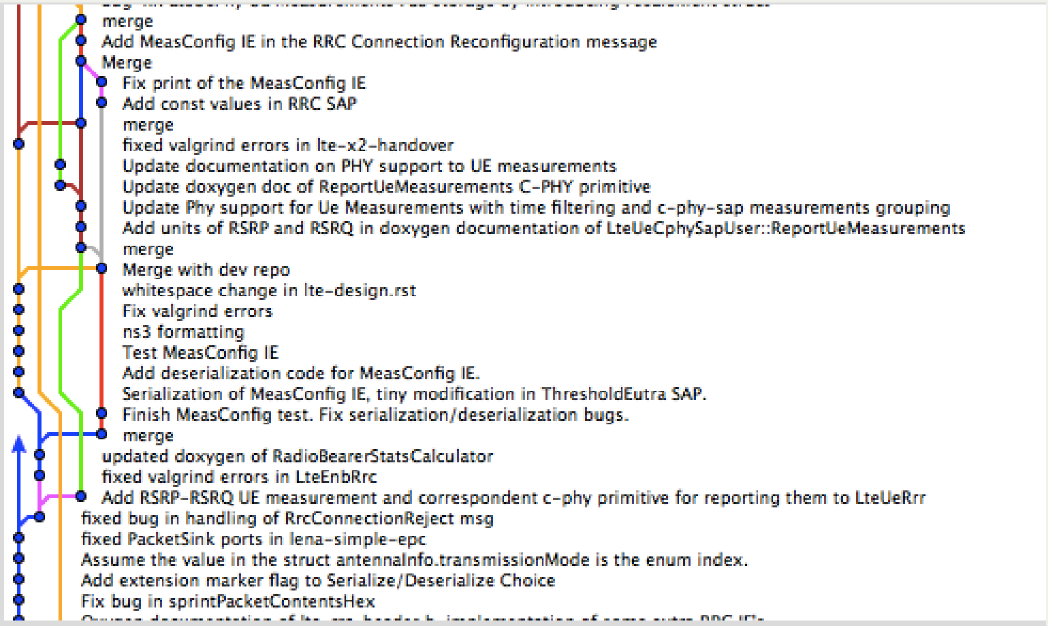
Recap

- Successful application of checkpoint/restore methodology to ns-3
- Userspace-only approach very appealing
- Suggest as a potential tool for saving time
- Many other potential applications, as previously documented in the literature

Operating Systems

- ns-3 is at least partially supported on Linux, FreeBSD, Mac OS X, it
- DMTCP is Linux-only
  - Reasons why here: http://www.slideshare.net/yuliang_neu/porting-dmtcp-macslides

Revision control for checkpoints?



Checkpoint Management for Interactive Simulation

- Revision control of potentially large files (BUP, git-annex, BOAR, git-bigfiles)
- Metadata
- Incremental checkpointing
- Files which do not remain open consistently (open/write/close)

Simulation Cloning

An eventual goal of this research is a generic, straightforward implementation of simulation cloning and/or other advanced simulation techniques which are easily applicable to a wide variety of simulations incuding ns-3

References

- https://github.com/kwharrigan/ns-3-dev-git/tree/checkpointer (the code)
- J. Ansel, K. Arya, and G. Cooperman. Dmtcp: Transparent checkpointing for cluster computations and the desktop. In Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, pages 1-12, 2009
- See timeline
- See paper for the rest

Questions / Feedback?