ns-3 developer meeting notes

May 8-9, 2014, Atlanta GA

Attendees: Budiarto Herman, Ken Renard, Jared Ivey, Peter Barnes, Nicola Baldo, Daniel Lertpratchya, Felipe Perrone, John Abraham, Cristiano Tapparello, George Riley, Tom Henderson, Brian Swenson, Tommaso Pecorella (virtual)

Topics:

- Generalized Wireshark filter capability for packets
- How to use ns-3 as a traffic generator (triggered events)?
- Inserting replacement protocol layers
- ns-3 Windows port
- Support for configuration hints for command-line configuration strings
- Documenting trace source callbacks
- Data Collection Framework work at Magister
- bake modularization next steps
- Object Start/Stop progress
- Memory scaling
- Unmaintained models

Generalized Wireshark filter capability for packets

John Abraham mentioned that he has a need to filter packets being saved to netanim trace files for large animations, and was thinking of supporting a Wireshark filter syntax (e.g. “tcp.port==80”) for users to specify filter combinations. John wanted to know whether this was a generic need for other ns-3 components or just a netanim need. Ken Renard suggested that libpcap and tcpdump (tcpdump with its own similar filter syntax) might be portable to ns-3, and there was brief discussion about whether pcap writing itself (for Pcap trace helpers) could use these libraries (one reason they were not originally was to avoid the library dependence). We discussed the possible need to otherwise filter things like ascii and pcap traces, and Nicola reminded the meeting that it was an early design decision not to build a complicated filter system for pcap and ascii traces, given that there exist many tools for post-processing the pcap output, to even filter it to ascii, so there was not much support for additional filters. However, netanim does have some need for this because it uses XML-based text, more verbose than binary pcap, and the meeting concluded that this was probably a netanim-specific need.

How to use ns-3 as a traffic generator

John mentioned the interest to use ns-3 as a possible traffic generator, but not one that operated independently but that could be driven by events in the external system. We had some discussion and John and George took the action item to look for past code (WNS3 2012 paper) regarding how to use ns-3 over real sockets, and make that available in a code review.

Inserting replacement protocol layers

John also mentioned that he had seen several industrial use cases where vendors want to be able to rip out parts of the protocol stack (e.g. everything above IP) and insert their own code, and we did not have much guidance or help to do that. We discussed this a bit and some people felt that there was not much more that ns-3 could do since we already have made the layering based on callbacks (which could support other stacks) and that needs from company to company to cradle their own code were not likely generalizable.

ns-3 Windows port

We asked about the status of the Windows port that John Abraham and Brian Swenson had made for ns-3.18, and the prospects (and costs) for long-term maintenance. The code review issue is located here:

https://codereview.appspot.com/download/issue8932044_1.diff

Some notable aspects of it are:

- it does not support Python at all, or some libraries such as GSL
- because there is no Python, Brian wrote a separate test runner,
- there was some problem in the past in using the double version of large integers, although Peter suggested to look at the internal ns-3 Cairo solution (which John had stubbed out)

We had discussion about the Visual Studio version (2012, 2013) and the difficulty in supporting multiple, and the possibility that Microsoft may at any time end free availability of the version that we use (Visual Studio 2012).

We had some discussion and review of the code. John mentioned that the windows specific issues may diminish over time if ns-3 is clang-compliant.

We discussed whether to try to maintain it. We had discussed having the ns-3 Consortium try to raise funding to maintain it, but it has been difficult to try to make this happen. Several people in the room supported the notion of trying to make it more mainstream, but perhaps call it a “beta” to defuse maintenance questions until it is better understood. We also agreed that more regular testing and a buildslave would be needed.

We also discussed the possible benefit of requiring Python (and using it for e.g. the test runner) and John said he'd look into it.

The actions taken were:

- after ns-3.20, create a few new review issues (core, network, internet, + all rest), and try to merge (John, Brian)
- make a new ns-3.20 “beta” release (John, Brian)
- obtain a Windows buildslave (Tom)
- create repository for project files (John)
- figure out impact on pending modularity (bake)
- open issue: whether to move towards Python support (John to look at)
- in the existing patch, where there are multi-line preprocessor statements, add macros for NS_WINDOWS_NO_INTERFACE(), or move them to a core imported header

Support for configuration hints for command-line configuration strings

Peter explained that when using the command-line argument system to select values, and incorrect values are provided, the program crashes with unhelpful output. For instance, “./waf –run “seventh –SimulatorImplementationType=ns3::Default?” could instead provide helpful output such as “ns3::Default not found, available choices are...”.

Peter explained why the programs crash and suggested two possible fixes extending the TypeId system, including a list string checker that enumerates (hard codes) allowable choices, and a base class string checker that uses a TypeId::GetChildren() method to allow the system to find the base and derived TypeIds that are legal for the argument. Peter was encouraged to try the base class string checker idea.

Documenting trace source callbacks

Peter mentioned that a weakness in current documentation is that trace source callback signatures are not documented. He proposed a solution whereby trace source callbacks should be typedeffed, and the typeId code for trace sources extended to store this

typedef. He gave an example of what he meant, and the proposal was slightly adjusted after discussion. Peter cautioned that this could require a lot of grunt work to annotate all of the existing trace sources. Peter will follow up on this.

Data Collection Framework work at Magister

Budiarto Herman described some work performed at Magister, in the context of the satellite model, on the Data Collection Framework (DCF). In particular, Magister has implemented several statistical collectors to fill in for those that are missing from the ns-3.18 release of the DCF, and reviewed how they worked (source code is not yet released). This prompted some discussion about the request from Nicola to consider replacing the file handling code in the LTE statistics helpers with DCF aggregators, and to consider the multi-valued tuples emitted by LTE trace sources (and how to handle them). Some additional discussion revolved around the tradeoff between configuration flexibility, data provenance issues, and complexity for the user. Felipe and Tom agreed to revisit the questions that Nicola raised again, and Budiarto agreed to share the Magister collectors at a future date.

bake modularization next steps

Tom Henderson reviewed the current capability of bake as a build tool for ns-3 modularization, and led a discussion about design choices. He plans to work on this refactoring over the coming year.

We discussed in the context of possibly factoring out a large module such as wimax, and the pros and cons related to building/maintaining such modules at the current ns-3-dev level or at the bake level in parallel with other libraries such as NSC and click. Again, it seems that it may be useful to consider splitting out all modules to coexist in parallel, if revision history on the modules can be maintained.

It was noted that this will be a painful process with respect to how examples currently are built, the documentation building frameworks, and the documentation (tutorials) itself. An incremental path is likely needed whereby bake and other scripts are extended with one or two new modules built in the new format (instead of merging them to ns-3-dev) and then once that approach is stable, to try to break apart the repository.

One open issue is whether it would be possible to remove directories from the repository and still retain revision history on the remaining. Peter remarked that mercurial tools may be able to do this. "hg convert --filemap" is supposed to enable splitting a repo, preserving history on each component.

We discussed whether there was interest in converting our repositories to git (if the history issue could be solved), and there was little support. We discussed GitHub as a possible public repo hosting solution. A Mercurial-friendly alternative could be Bitbucket. Either opens up the possibility of using their integrated issue trackers, and code commenting/review.

Object start/stop progress

We reviewed Vedran Miletic's latest patch for providing Failure/Repair capabilities to Objects, and the wiki page concerning the design goals and previous agreements.

Questions again arose as to what we are trying to model. We list below a few illustrative use-cases that were discussed:

- rover vehicle stops when batteries drained, reboots (or wakes) when solar kicks in: mobility model stops in this case
- satellite sleeps/wakes: mobility continues while asleep
- excluded volume mobility models need to know there is a dead object, so you can't just delete a non-restarting Node and all its contents. It needs to preserve its position.

Peter emphasized the clarity that a state diagram might lend to the discussion. We reached consensus on one such diagram (attached). It seems like we need to distinguish between Stop/Restart and Sleep/Wake, and probably provide these as separate capabilities (that may be combined).

In this approach, Sleep/Wake would probably be used in energy-aware contexts, and would mainly correspond to the behavior that event processing (packet reception and generation) would be suspended while in a sleep state, but most object state would be preserved in the transition. However, Stop/Restart would restore the object to its original post-construction state, and some objects may just be stoppable and not restartable. If an object were to provide Restart(), it would need to do some additional work to make sure that all post-construction state was cached upon first initialization.

We discussed the implementation approach. In order to optionally provide this such that the trait is queryable via GetObject(), there seem to be two approaches:

1) use the mixin approach but extend the Object::DoGetObject to search a directed graph structure and not a tree structure when it searches for TypeId matches
2) else provide a set of new base class objects such as “StoppableObject”, “ResumableObject”, and “StoppableAndResumableObject” that mixed in the

appropriate interfaces

We suggest that 2) would be preferable to avoid TypeId changes. These classes StartStopFunctionality and SleepWakeFunctionality could also be used (mixed in) for non ns3::Object classes if desired.

We had some discussion about the difficulty in automating the stopping of associated objects, such as whether calling stop on a Node should stop its motion or not (it may depend). We initially considered that, we ought to focus on just stopping objects individually and not trying to have the Stop() propagate to other aggregated objects. However, in the example use case of a Node running out of energy then later awakening, we may need to in fact to provide a Node-level Sleep/Wake that calls into the constituent objects. So we concluded that we ought to perform a node-level example.

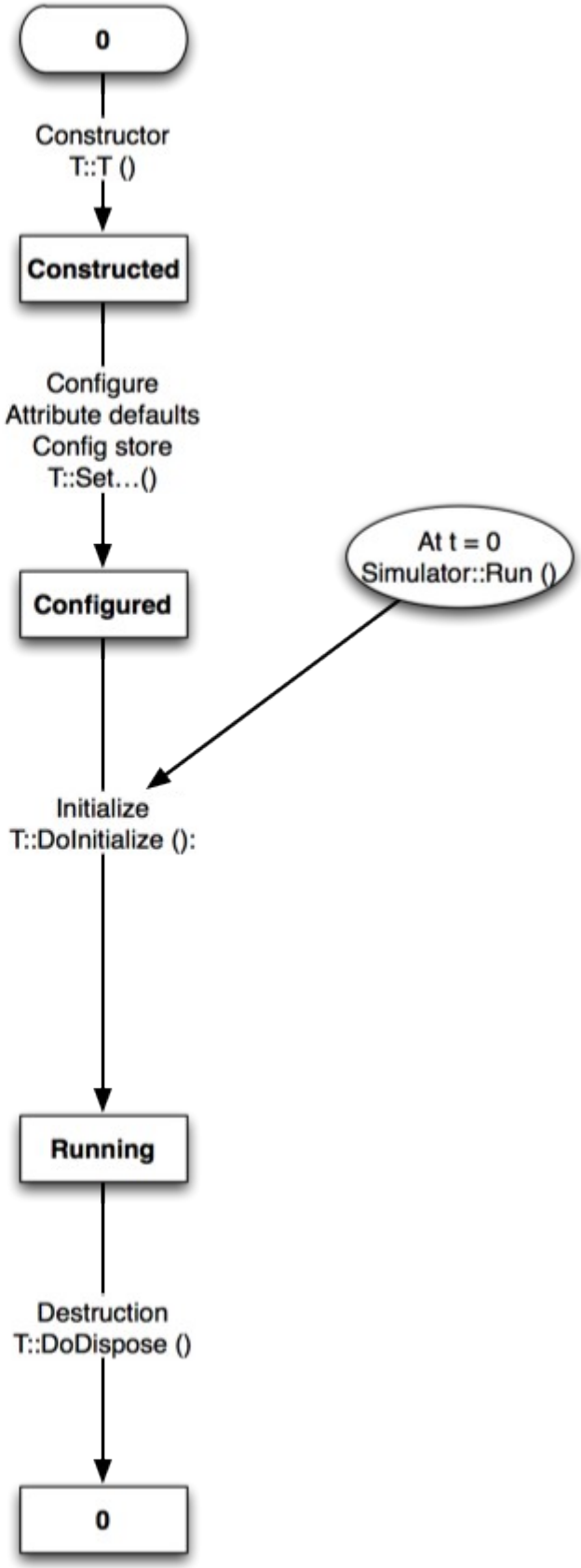
We discussed specifically that handling of Sleep and Stop events may need to be handled in different ways by any mobility models; there may be some scenarios where energy depletion results in motion pausing or continuing, depending on the desired effect. So, it is not necessarily the case that Node::Stop() or Node::Sleep() should cause those operations to occur on the mobility model; we'll need to be able to handle different policies. This affects the previously-held assumption that Node::Stop() should stop all aggregated objects.

Suggested next steps:

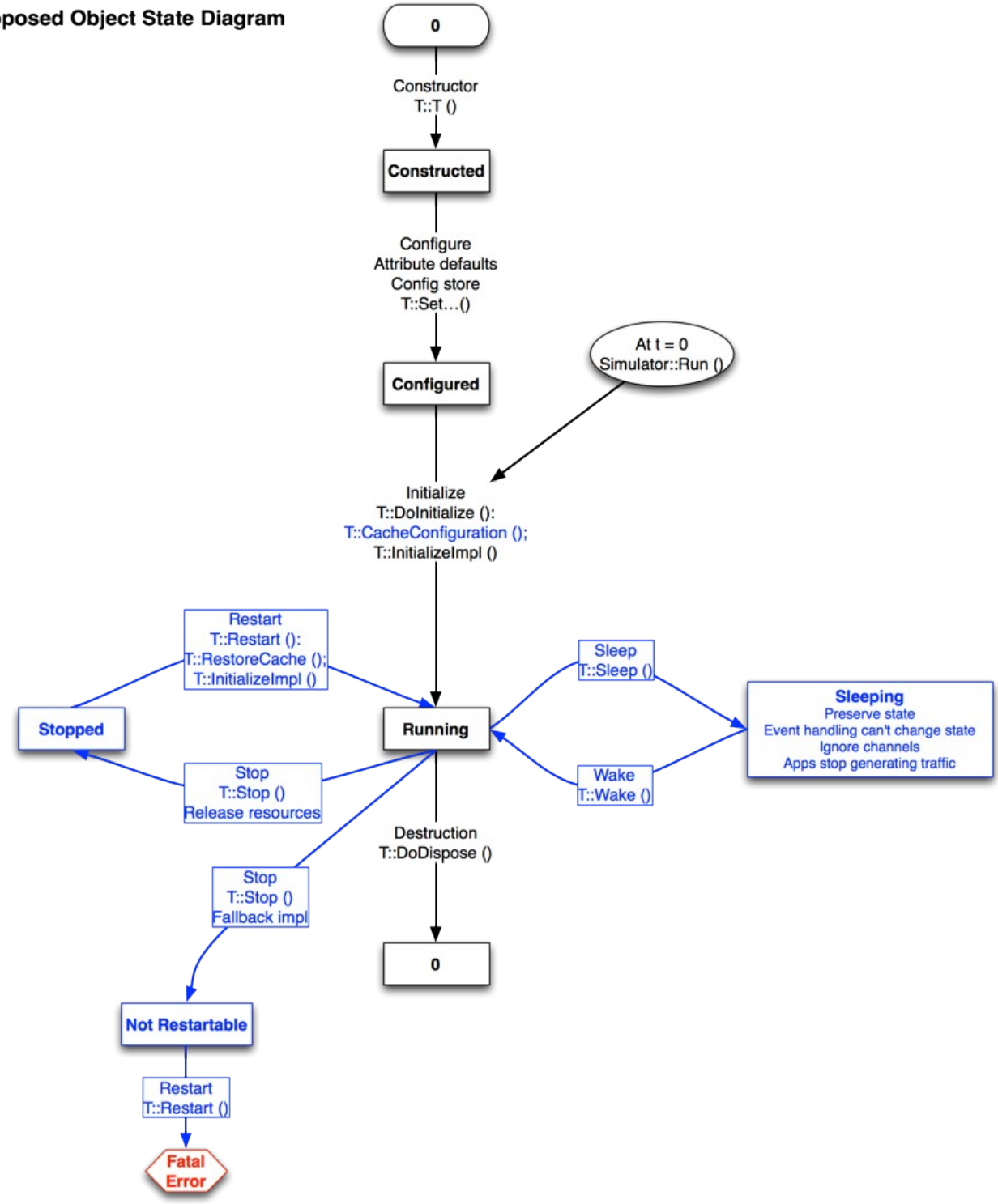
1) implement test objects to instantiate
2) try to implement Sleep/Wake example of a Node running out of energy, then later getting more energy, and doing something sensible. Either WiFi or LrWpan devices are probably the best ones to attempt this with.

A final note was that there were some comments in the wiki noting the need to call "ScheduleWithContext()" upon restart events, and people were unaware of the status of the threaded scheduler, so we agreed to look into this.

Current Object State Diagram



Proposed Object State Diagram



Memory scaling

Peter Barnes reported on some work that he has been involved with involving memory scaling. The distributed simulation implementation requires instantiating a ghost node in every rank, which results in a limit in maximum model size. For instance, 32 GB of RAM is restricted to 750,000 nodes, so no matter how many servers exist in the cluster, the number of nodes is restricted by the amount of memory on a single server. Peter described some work started to eliminate the need to instantiate ghost nodes on other ranks. The routing frameworks (Nix vector and global) need adaptation or replacement, and some ideas are in work, possibly including Boost Graph Library. The requirements are that 1) distributed simulations produce identical results with sequential simulations, and 2) results do not vary depending on how topology partitions are made.

Peter also mentioned that work on XML-based scenario description (originally mentioned at WNS3 2013) was ongoing, and some discussion ensued about the relationship of this work to other related work on XML-based descriptions.

Unmaintained models

Tom reported that the project has lost some maintainers and we have some several longstanding bugs or patches pending which are stalled due to lack of maintainer. Nicola suggested that we actively try to plug the gap, and we were able to identify a few people on the spot who could help (Cristiano for energy models, Nicola for propagation models, Jared for mobility models). Peter suggested to use the bug tracker to generate reports of open bugs, which suggested that the unmaintained modules were not that problematic compared to those of selected maintained modules (core, internet).