Integrating ns-3 Model Construction, Description, Preprocessing, Execution, and Visualization

Workshop on ns-3 WNS3 2013
Cannes, France  March 5, 2013

Peter D. Barnes, Jr,
Betty B. Abelev, Eddy Banks, James M. Brase
David R. Jefferson, Sergei Nikolaev, Steve Smith, Ron A. Soltz

LLNL-PRES-622874
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, Lawrence Livermore National Security, LLC
Simulating Large Networks

- dB → 20k node model
- First attempt: direct export to C++
  - Multi-MB source file
  - 10’s minutes to compile
- Export to XML intermediate
  - Compile-once executable
  - Seconds to parse and instantiate model
Describing a Model is Hard

- Topology, addresses
- Channels: bandwidth, latency, error rates, …
  - Environmental: noise, interference, multi-path
- Framing, access control, devices
- Packet-level protocols, congestion control, queuing discipline, buffer sizes
- Routing!
- Traffic generators/consumers (applications)
- Measurements: parameters, statistical reductions, recording intervals
- Parallelization, checkpoint/restart, dynamic load balancing
- Parameter studies
Tension Between Framework-Agnostic (Generic) vs. -Specific

- Usable by multiple frameworks
- Larger potential user/developer base
- Less tailored to any one framework
- Difficulty mapping concepts
- Implementation requires broad knowledge
- Cross-community communication issues
- Tailored to one target
- Use familiar terms
- Smaller set of use-cases
- Tighter community buy-in
- Leverage existing code base
- Smaller user/developer community
- Models not portable between frameworks
Generic \leftrightarrow Specific Dimensions

- **Concepts**
  - Explicit: "NodeContainer"
  - Implicit: nodes on this CSMA
  - Tracing and output specs

- **Terms**
  - "NodeContainer", vs. "Network [...]"

- **Models**
  - Does shortest path routing mean
    - OpenFlow?
    - OSPF?
    - BGP-nsc? -dce?
    - Native BGP?
    - nix-vector?
    - GOD-routing
Abstracting Models from Simulators Has Wide-Ranging Benefits

- **Specificity**
  - Model description not obscured (or less obscured) by implementation details

- **Portability**
  - Here’s my model, run it in your simulator
  - Explore new model elements with limited availability

- **Comparison of simulation frameworks**
  - Portable performance benchmark models
  - Cross-validation of results
Non-Simulation Benefits

- Enable multiple visualization options
  - Write visualizer once, use with many simulators

- Portable model generators
  - Graphs, RocketFuel, CAIDA, …
  - Generate standard model file readable by all simulators
  - (Instead of porting the generator to all simulators)

- Standardized parallelization tools
  - Partitioning, weighting
Various Approaches to Model Specification

- **Source code**
  - ns, ns-2, ns-3, ROSSNet

- **Private text formats**
  - ~INI file: SSFNet, REAL, …
  - XML: NEDL/NSTL, EMANE, NetDMF/XDMF
  - Other: QualNet

- **Private languages**
  - OmNet++ NED

- **Proprietary, binary**
  - OpNet
Prior Work Abundant – Notable Features – I

- Disclaimer: errors are my own
- OpNet
  - GUI-driven, proprietary binary file format
- SSFNet
  - Extensible grammar,
  - Elements of inheritance, composability and reuse
- OmNet++
  - NED-grammar compiles to C++,
  - Run-time configurable elements
  - Visualizer annotations included
Prior Work
Notable Features – II

- **eMANE**
  - Well-defined XML schemas
    - `<name, value>` pairs
    - Limited inheritance for alternate parameter values

- **NetDMF/XDMF**
  - Automatic conversion between light data (directly in XML) and heavy data (external dB, HDF5)
  - `<name, value>` pairs
  - Need translators to each target environment (simulation, emulation)

- **ns-3**
  - NEDL/NSTL – parametrized and templated models, predefined by instructor
  - ns3xml – moderately detailed grammar, somewhat rigid implementation, mixes visualization with model
Desirable Feature Highlights – Grammar

- (See the paper for a more complete list)
- Complete
  - Capture entire model in a unified grammar, in a single file
  - Including measurement points and reductions
- Validated
  - Against the fully specified grammar
- Canonical
  - Transform to canonical form, enables model comparison/differencing
Desirable Feature Highlights – Implementation

- Leverage common libraries to ease parser construction

- Modular grammar
  - Ease creation of XSD and parser for new model elements
  - `<name, value>` generics for elements without XSD

- Support external configurations
  - Protocol/device configuration files
  - Virtual machines
Desirable Feature Highlights – User Interaction

- Recognizable grammar, human and machine readable
- Composable/hierarchical/parameterized
  - Reuse: models can be incorporated as sub-elements in larger models
  - With parameter substitution (base addresses, …)
- Specialization
  - Explicit control of generic → specific
XML Network Description Language (XNDL) Design Goals

- XML, with complete XSD grammar
- Familiar networking terms
- Compact
- Simplify with reuse
- Separate model from meta-data
- Simulator-agnostic tools
- Use XSLT to transform
Examples: Parameter Study, Visualizer

- Start with fully-specified model
- Use XSLT to apply specific parameters, generating test points
- Mix-in visualizer hints
Model Portability

- Use XSLT to transform between generic and specific grammars
  - Element interpretation explicit, under modeler’s control
ns-3 XNDL Ecosystem

Model Combiner

Web Frontend to Cluster
- NEDL/NSTL

Generic Model Generators
- NetworkX
- Examples
- Flat Benchmark

Custom Models
- LLNL
- ARL

Visualization
- NetAnim
- LLNL Everest

Partition
- METIS/pMETIS
- Scotch/PT-Scotch

Global XML File
- model-all.xndl

Per-Rank XNDL Files
- model.r0.xndl
- model.r1.xndl

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAnim
- LLNL Everest

Visualization
- NetAn
#include "ns3.h"

int main(int argc, char **argv) {
    std::string xmlFileName;

    CommandLine cmd;
    cmd.AddValue("xmlFileName", "Path to XML file", xmlFileName);
    cmd.Parse(argc, argv);

    XMLSimulation simulation (xmlFileName);

    simulation.Run();
    simulation.Report(std::cout);

    return 0;
}
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Simulation XML file -->
<XNDL
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="XNDL.xsd"
    SchemaVersion="1.0"

    <!-- Model name -->
    Name="Campus Network v2.9"

    <!-- Anything can have a description -->
    <Description>
        This model describes ...
    </Description>

    <!-- Global attributes -->
    CsmaEnableAsciiTraceAll ="false"
    CsmaEnablePcapAll ="false"
    P2pEnableAsciiTraceAll ="false"
    P2pEnablePcapAll ="false"

    <!-- Override on command line -->
    RandomSeed = ...
>

XNDL
Ref: Inheritance/Reuse

<!-- CSMA channel template -->
<Subnet Name="csma"
    DataRate="100Mbps"
    Delay="1ms" >
</>

<Subnet Name="Building 1-3rd Floor"
    <!-- Reuse the "csma" parameters defined above -->
    Ref="csma"

    <!-- But change this parameter -->
    Delay="2ms"

    <!-- Add new nodes to the model -->
    Size=20

    <!-- Automatic parameter assignment where reasonable -->
    Cidr="10.12.133.64/27"

/>
XNDL
Ref and Index

<Subnet Name="Building 1–Building 2"

<!-- Use the previously specified point-to-point parameters -->
Ref="p2p"

<!– Install P2P on nodes instantiated elsewhere -->
<Node Ref="Building 1-3rd Floor"
   Index=3

   <!-- Explicit address assignment -->
IPv4="192.10.1.40"
   DNS="rtr-b1-f3-00"
/>

...

</Subnet>
XNDL Applications

<Application Name="Admin-browser"
    Type="OnOffApplication">
    <Parameters>
        <DataRate>100000</DataRate>
        <OnTime>2</OnTime>
        <OffTime>20</OffTime>
    </Parameters>
</Application>

<Application Name="Admin-email" ... />

<ApplicationSet Name="Admin">
    <Application name="Admin-browser" />
    <Application name="Admin-email" />
</ApplicationSet>

<Subnet Name="Front Office" />

<!-- Install these apps randomly to 80% of the nodes -->
<ApplicationSet Ref="Admin" Percentage="80"/>

</Subnet>
XNDL

Generic Applications

```xml
<Application Name="GenericOnOffTest"
  Type="GenericApp">
  <Parameters
    <!-- ns-3 TypeId -->
    Type="ns3::OnOffApplication"
    <!-- Base class parameters -->
    Start="2.0" Stop="20.0">
      <!-- Name, value attributes -->
      <Attribute Name="DataRate" Value="500bps"/>
      <Attribute Name="PacketSize" Value="50"/>
      <Attribute Name="Remote" Value="03-06-0a:01:03:0f:50:00"/>
  </Parameters>
</Application>
```

- Uses ns-3 Type and Attribute systems
  - Create the object from a factory, obtaining the c’tor from `TypeId::LookupByName ()`
  - Configure attributes from `<name, value>` pairs
    - Assumes (as does ns-3) that attributes are representable as strings
Automatic Partitioning Process

- **Sector tool**
  - Labels unpartitionable regions (e.g. CSMA channels)
  - Generates weighted sector graph

- **METIS (or Scotch)**
  - Selects rank assignment for each sector, minimizing global penalty function

- **ApplyRanks XSLT transform**
  - Apply METIS assignments, generating (per-rank) partitioned model file(s)
Automatic Partitioning
Advantages

- Sector labeling performed once
- Simplifies exploration of different partitioners, weighting
  - Rerun partitioning step with different arguments
- ApplyRanks can produce
  - A single .xndl model file
  - An .xndl file per rank, or group of ranks (expected to improve parallel I/O for very large models)
- XSLT preprocess step to improve parallel performance
  - Replace 2-node CSMA with P2P (correcting bitrate for duplex → simplex)
  - Refactor high-centrality routers as partitionable cliques
Limitations

- Is one .xndl the complete model? – NO!
- Some things obviously should be included by reference:
  - VM images
  - Models with well-established (real-world) external configuration schemes
    - Quagga BGP, ...
- Gray areas
  - Phenomenological probability distributions
Still To-Do

- Complete coverage of ns-3 models
  - Wireless
  - Position, mobility
  - Buildings, antenna
  - Support Tracing and output
- Output .xndl at Simulator::Run ()
- Support for NetAnim uses