Pushing the Envelope in Distributed ns-3 Simulations: The Quest for One Billion Node Simulation

WNS3 2015, Castelldefels, Spain
May 13, 2015

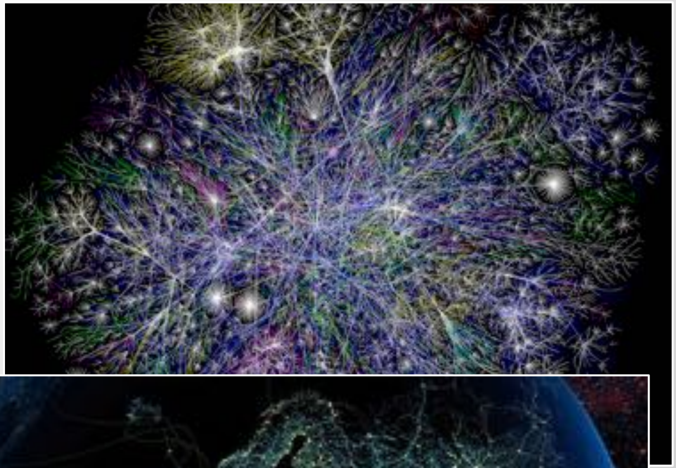
S. Nikolaev, L. E. Banks, P. D. Barnes, Jr., D. R. Jefferson, S. G. Smith

Lawrence Livermore National Laboratory

LLNL-PRES-671055
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

Overview

- Focus: distributed ns3 simulations, parallel scheduler, model-building tools
- Objectives
  - Develop custom network model-building tools to enable simulation of very large computer networks
  - Study the scalability of distributed ns-3 in terms of the traffic runtime and memory footprint by varying CPU ranks and the size of the model
  - Compare performance of two parallel schedulers, YAWNS and NULL



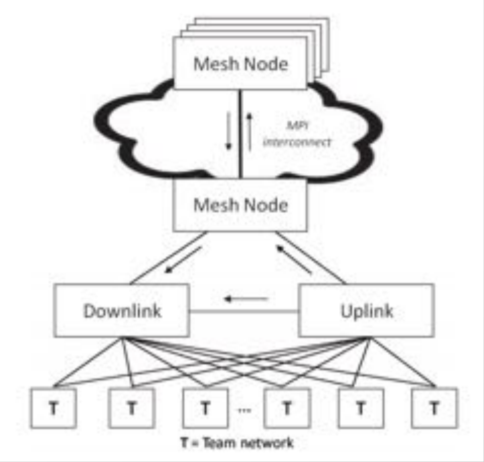
Can we model planetary-scale (~1B nodes) networks using standard ns-3 today?

Rank	Country/entity	IPv4 addresses	%
	World	4,294,967,296	100.0
1	United States	1,541,605,760	35.9
	Bogons	875,310,464	20.4
2	China	330,321,408	7.7
3	Japan	202,183,168	4.7
4	United Kingdom	123,500,144	2.9

Source: http://en.wikipedia.org/wiki/List_of_countries_by_IPv4_address_allocation

Previous Work

- E. Weingärtner, H. vom Lehn, and K. Wehrle (2009)
  A performance comparison of recent network simulators
- J. Pelkey and G. Riley (2011)
  Distributed simulation with MPI in ns-3
- P. D. Barnes, Jr., *et al*. (2012)
  A benchmark model for parallel ns3
- K. Renard, C. Peri, and J. Clarke (2012)
  A performance and scalability evaluation of the ns-3 distributed scheduler
  - 360M network nodes!



- S. Nikolaev, *et al*. (2013)
  Performance of distributed ns-3 network simulator
  - Scalability study, performance metrics (runtime, RSS footprint), XNDL
  - Largest model: 750K network nodes (node RAM-limited)

- S. G. Smith, et al. (2015), WNS3 2015
  Improving Per Processor Memory Use of ns-3 to Enable Large Scale Simulations

Enabling Developments since SIMUTools'13

Distributed network topology

Instead of keeping the entire network model in memory, each processing rank stores only local (subset) network. This allows to scale to the cluster memory instead of the compute node memory.

XNDL maturation

XML Network Description Language development specifically targeted very large networks.

New distributed scheduler

A new distributed scheduler is based on NULL message exchange and can lead to performance improvements for certain types of modeled networks.

Incremental ns-3 development

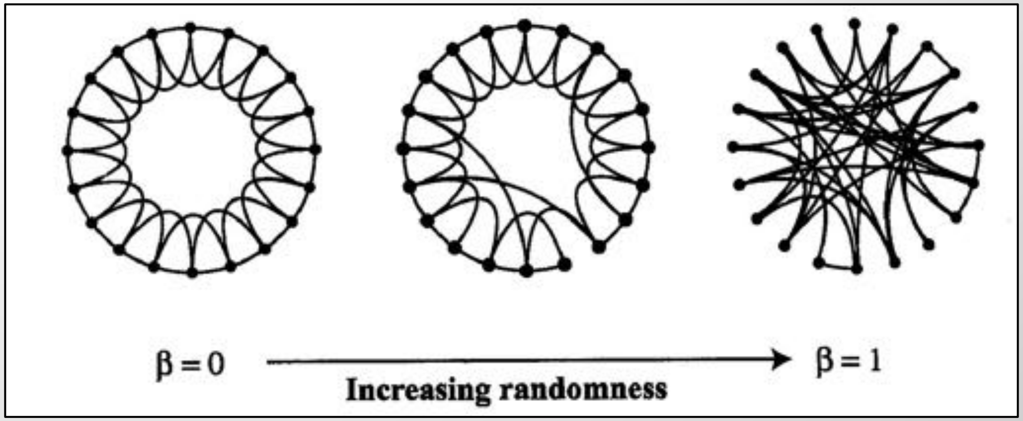
New version of ns-3 include performance improvements and optimizations that may enable running larger models.

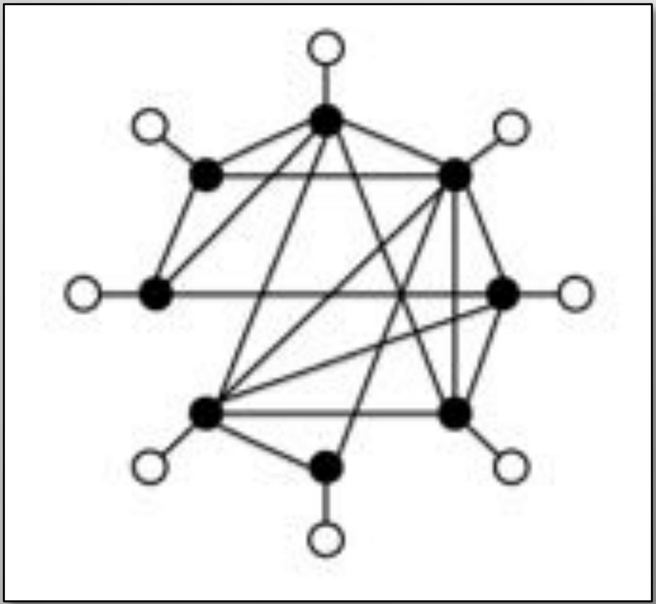
Evolution of computing hardware

A new cluster, *catalyst*, became available with large RAM (128GB/node; 324 nodes) and NVRAM (800GB/node). Another cluster, *herd*, with very large RAM (1024GB/node; 9 nodes) for memory-intensive processing.

Simulated Networks

- Small-world network of routers
- (k=4, β=0.5)
- Watts-Strogatz algorithm
- Mimics the backbone/AS
- Each router is connected to a single leaf node





For ***N*** total nodes:

N/2 routers

N/2 leaf nodes

N/2 (CSMA) +*Nk/4* (P2P) = ***3N/2*** total channels

Range of models simulated:

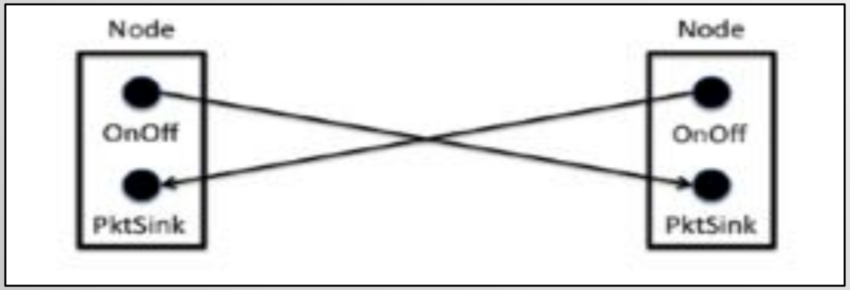
2^{17} → 2^{30}

131,072 → 1,073,741,824

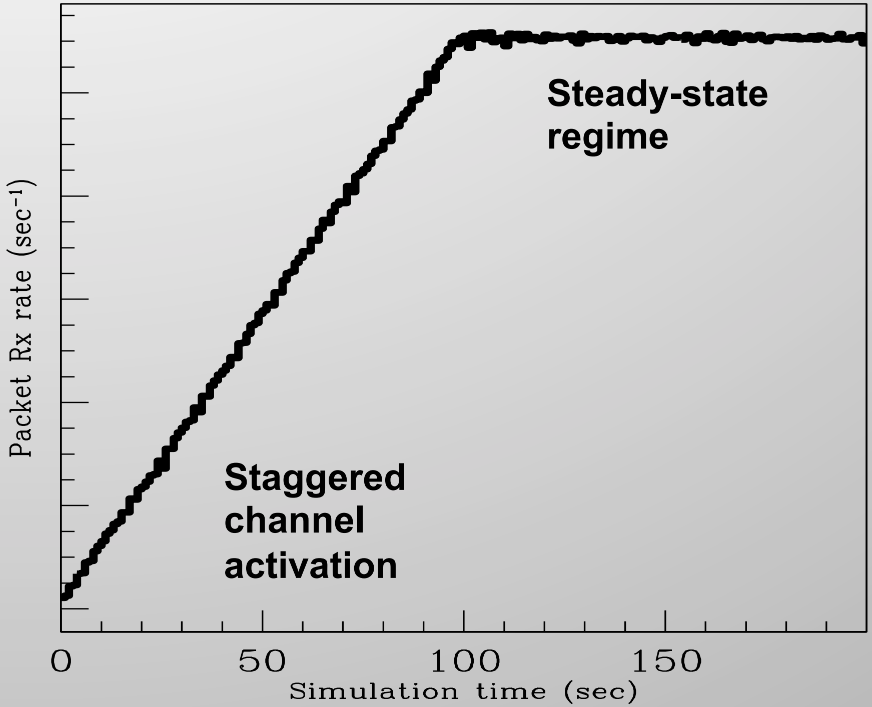
Since each channel has 2 unique IP addresses, the total address space is 3N.
For 2^{30} model networks, we are approaching the limit of the IPv4 address space.

Network Traffic

- Full load models: Packet traffic is generated on *every* channel
- Single-hop routing: to avoid routing artifacts, only consider traffic between immediate neighbors
- Each channel has two OnOff Apps, cross-wired to two PacketSinks



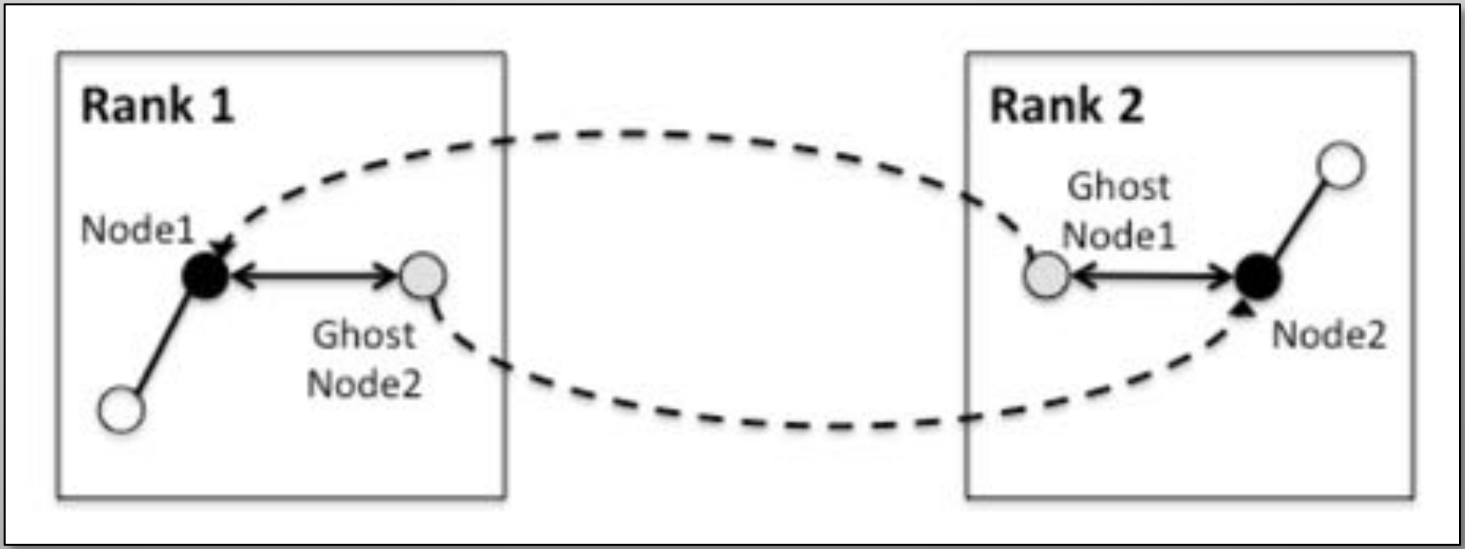

```
Packet size: 500 bytes
Protocol: TCP
Data rate: 5000 bps
Link delays are U in [2,12] ms
Channel bandwidth: 100 Mbps
Mean packet rate: 0.56 pps
Mean arrival interval: ~1.8 sec
```



Simulation runs for 200 sim seconds: 100 (channel activation) + 100 (steady-state). To isolate the effects of routing calculations, channels activate randomly during the first 100 simulation seconds (staggered activation). The second 100-sec interval is ‘steady-state’ regime.

Distributed Topology

- With distributed network topology, each MPI ranks only needs to store the part of network it is modeling (plus the neighbor nodes); before each MPI rank had to store the entire network topology (node RAM bottleneck)
- To distribute, the network is partitioned into *sectors*
  - sector must reside on a single MPI rank; cannot be split
  - since splitting is only allows across P2P channels, sector = CSMA subnet
- To enable communications among the partitions, we introduce *ghost nodes*
  - ghost nodes are created during model initialization
  - send and receive MPI communications between the ranks
  - handshake between ghost nodes and underlying real nodes to set up MPI links



Schedulers

- YAWNS scheduler
  - default ns-3 distributed scheduler
  - look ahead time, synchronization phase
  - global mechanism; may result in performance hits for very large number of ranks
- NULL scheduler
  - pair-wise (not global); expect good performance for sparse rank graphs
  - packets continuously update look-ahead
  - overhead of nulls only incurred when packets aren't frequent enough between a pair of ranks
  - explicit NULL messages exchanged between ranks, giving the look-ahead information (e.g. '*don't expect anything from me until time X*')
  - avoids deadlocks

The choice of a particular scheduler is done at runtime, through a command-line option

XNDL Features

XNDL Features	Description / Benefit
Separability	Rather than specifying the model as part of ns-3 simulator using C++ syntax, the model is completely detached from the simulator ➢ Compile-once: Do not need to recompile ns-3 every time the model is changed ➢ The model code is drastically simplified ➢ Model description not obscured (or less obscured) by implementation details
Portability	Facilitates model sharing (e.g. "*here's my model, run it in your simulator*") XML file with complete XSD grammar (free error-checking) Allows comparison of simulators and cross-validation of simulation results
Modularity	Custom XSD and handler code lives in the ns-3 module directory Easy creation of XSD and parser for new model elements <name, value> generics for elements without XSD, leverages ObjectFactory, obtaining the c'tor from `TypeId::LookupByName ()` Configure attributes from <name, value> pairs Assumes (as does ns-3) that attributes are representable as strings
Familiarity	An XML file, so it is human- and machine-readable Recognizable grammar, uses familiar ns-3 elements e.g. *NodeContainer*, *Application*, etc Custom (intuitive) XSD for specific types

XNDL is an effort to design an XML-based language defined by a generic network domain model XSD

Compile-once XmlSim (entire ns-3 source file)

```
#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;
}
```

XNDL Features

XNDL Features	Description / Benefit
Compactness	Compact description Lightweight parser Support for compressed I/O streams XML compresses very well, ~95% compression
Hierarchical Composability	Existing models can be incorporated as sub-elements in larger models Facilitated by indirection (reference) in XNDL: elements are referenced by their (container,index)
Parameter substitution	Supports parameter substitution (e.g. base address)
Inheritance	Defines an element template using default parameter values Defines subsequent elements by referring to the parent element name and substitutes relevant parameters Leads to more compact model description

* Composability, Parameter Substitution and Inheritance are not fully operational

XNDL: XML Preamble

```
<?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”
  FileNumber              ="1"
  TotalFiles              ="2"

  <!-- Override on command line -->
  RandomSeed = …
/>
```

XNDL: NodeContainers and Subnets

```xml
<NodeContainer Size="10" Name="ALL_NODES"/>
<NodeContainer Size="100" Name="ALL_NEW_NODES"/>

...

<NodeContainer Name="p2p_12_nodes">
    <RefNode Name="ALL_NODES" Index="3"/>
    <RefNode Name="ALL_NODES" Index="0"/>
    <ApplicationSet Name="WebBrowsingSet7_0" Index="0"/>
    <ApplicationSet Name="WebBrowsingSet7_1" Index="1"/>
</NodeContainer>

...

<Subnet Cidr="1.0.0.22/31" Type="P2P" Name="p2p_12"
        NodeContainer="p2p_12_nodes" DataRate="100Mbps" Delay="10ms“>
    <Description>p2p_12_Subnet</Description>
    <RefNode Type="ROUTER" DnsName="router4.llnl.gov" Index="0“>
        <IPAddress>1.0.0.22</IPAddress>
        <MAC>00:00:00:00:00:16</MAC>
    </RefNode>
    <RefNode Type="ROUTER" DnsName="router1.llnl.gov" Index="1“>
        <IPAddress>1.0.0.23</IPAddress>
        <MAC>00:00:00:00:00:17</MAC>
    </RefNode>
</Subnet>
```

XNDL: ApplicationSets and Applications

```
<ApplicationSet Name="WebBrowsingSet7_0">
    <Application Name="PacketSink"/>
    <Application Name="Client_7"/>
</ApplicationSet>
<ApplicationSet Name="WebBrowsingSet7_1">
    <Application Name="PacketSink"/>
    <Application Name="Server_7"/>
</ApplicationSet>

...

<Application xsi:type="PacketSinkAppType" Name="PacketSink"
   Protocol="ns3::TcpSocketFactory" LocalAddress="0.0.0.0" LocalPort="80"
   Start="0.0" Stop="200"/>

...

<Application xsi:type="OnOffAppType" Name="Client_7"
   Protocol="ns3::TcpSocketFactory" DataRate="5000bps" PacketSize="500"
   RemoteAddress="1.0.0.23" Port="80" Start="94.4669" Stop="200"/>
<Application xsi:type="OnOffAppType" Name="Server_7"
   Protocol="ns3::TcpSocketFactory" DataRate="5000bps" PacketSize="500"
   RemoteAddress="1.0.0.22" Port="80" Start="94.4669" Stop="200"/>

...
```

Hardware

- *catalyst* cluster
- 149.3 TFLOP/s (theoretical peak)
- 324 nodes (7776 cores)
- Infiniband QDR (Qlogic) interconnect
- 41.5 TB total RAM
- 800GB local NVRAM per node
- Disk I/O 2PB capacity, 7GB/s bandwidth
- TOSS 2.2 (RHEL 6), mvapich2-gnu-2.0
- Each node is a pair of 2.4 GHz Intel Xeon E5-2695 v2 CPUs, 24 cores and 128GB DRAM per node

- *herd* cluster
- 1.6 TFLOP/s
- 9 nodes (256 cores)
- Infiniband QDR (Mellanox) interconnect
- 4.0 TB total RAM
- Disk I/O 5PB capacity, 7GB/s bandwidth
- TOSS 2.2 (RHEL 6), mvapich2-gnu-2.0

Both *herd* and *catalyst* have greater RAM per node than *cab* cluster used 2 years ago. With RAM as the bottleneck (even in distributed network topologies), our focus was on using clusters with large RAM per node.

Putting it all together: Making/Running a Network Model

Generate Model

Uses WS algorithm to generate basic graph structure given N. Writes the initial XNDL file with NodeContainers, Subnets, Applications, etc, but no sector information. Can be done in parallel, with each rank writing a subset of xndl files. Can also specify the number of output files.

Sector Labeling*

Initial xndl model files are analyzed to derive the sector information (CSMA networks). A sector must reside on a single MPI rank. Specifies potential partitioning, without actually partitioning (no ranks information). This needs to be done only once, regardless of the target number of ranks.

Model Partitioning

Given the target number of MPI ranks and the sector information from previous step, partition the model using graph partitioning (METIS). The xndl files now have all the information how to partition the model among the ranks; the information is mixed together.

Model Separation

Instead of having all MPI ranks read the same huge file(s), this step splits the original model file(s) into the chunks holding information relevant for target rank only. This drastically reduces the I/O.

ns-3 Run

Runs the network model, collecting performance metrics. Runtime choice of scheduler.

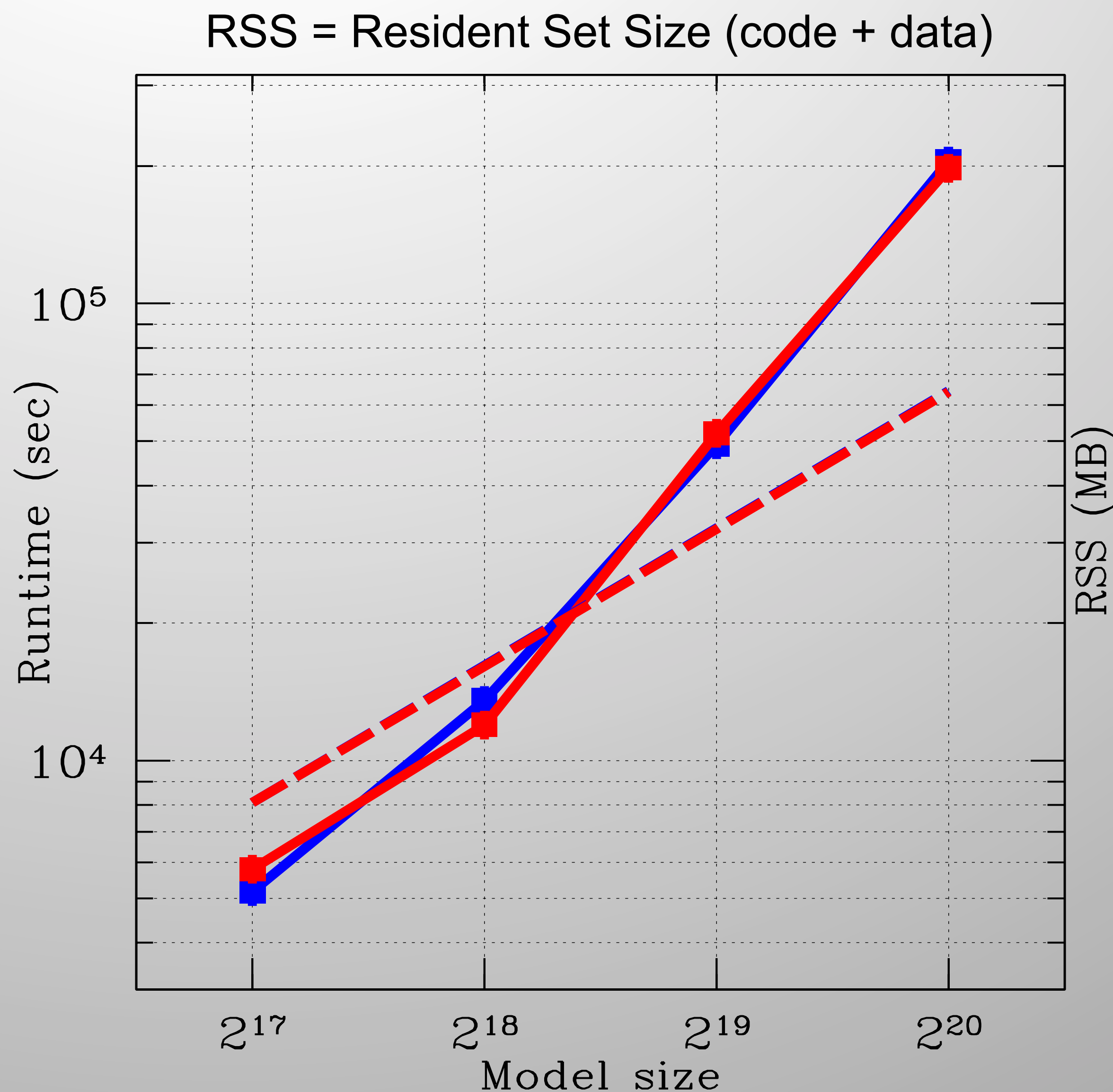
Model pipeline supports compressed streaming XML I/O throughout

Results

- Serial vs. Parallel-1
  - Any MPI overheads
  - Find the max size of a model that can fit onto a single node
- On-node vs. Off-node scaling
  - Difference in performance using cores vs. nodes
- Strong scaling
  - Holding the model size fixed, increase the number of MPI ranks for the run
- Weak scaling
  - Increase the number of MPI ranks while holding the job size *per rank* fixed
- YAWNS vs. NULL scheduler comparisons
- Metrics
  - Packet throughput rate
  - Runtime (total, setup, routing, traffic)
  - Memory (RSS) footprint

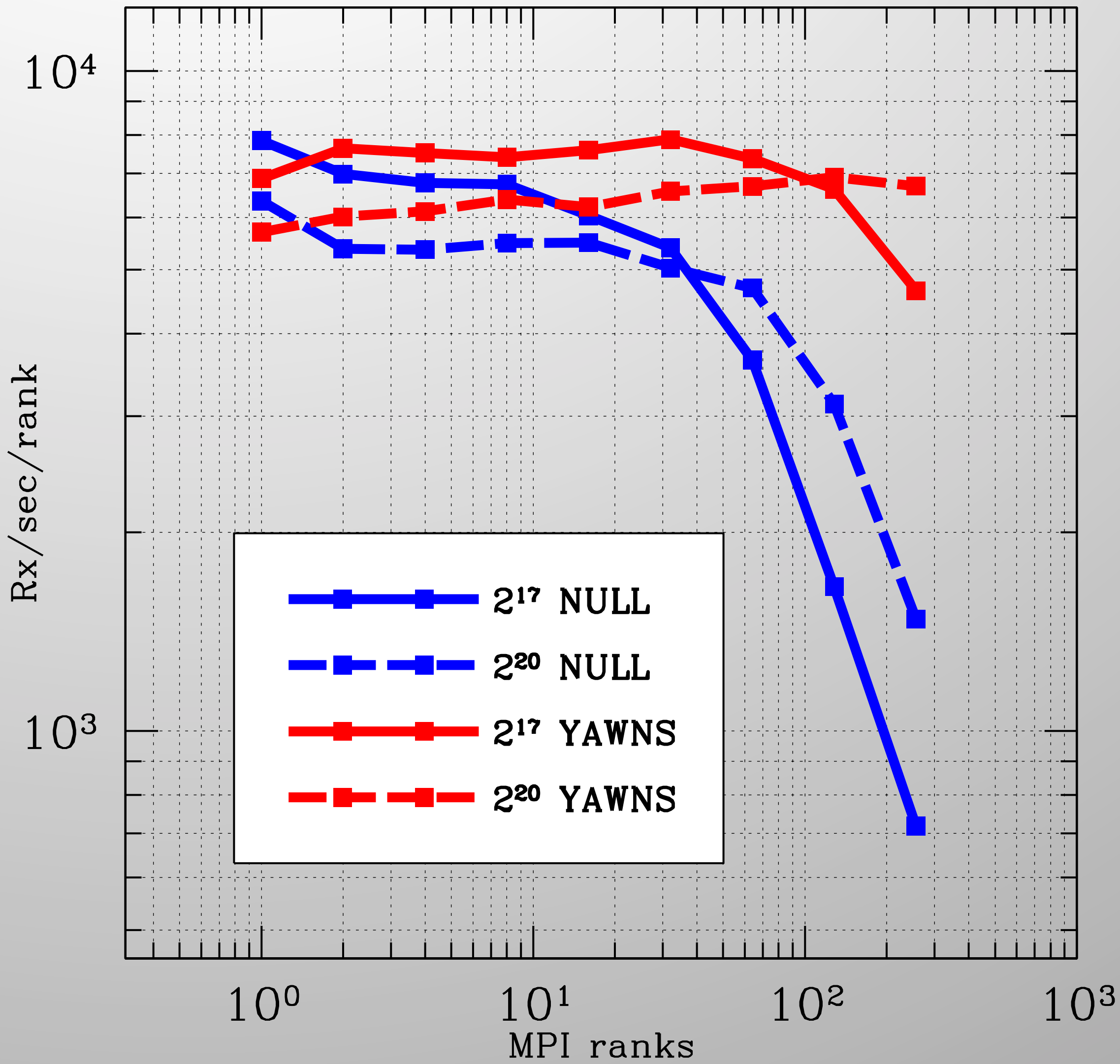
Serial vs. Parallel (1)

- Very little difference, no significant MPI overhead
- Typical runtime uncertainty is 5% (symbol size)
- Super-linear relation for runtimes (solid line) provided a motivation for the study in the companion paper
- Relation for RSS is linear (dashed line)
- The largest model to fit onto a node is 2^{20}; 2^{21} almost fits
- This limitation was due to inefficiencies in XNDL parser; now fixed

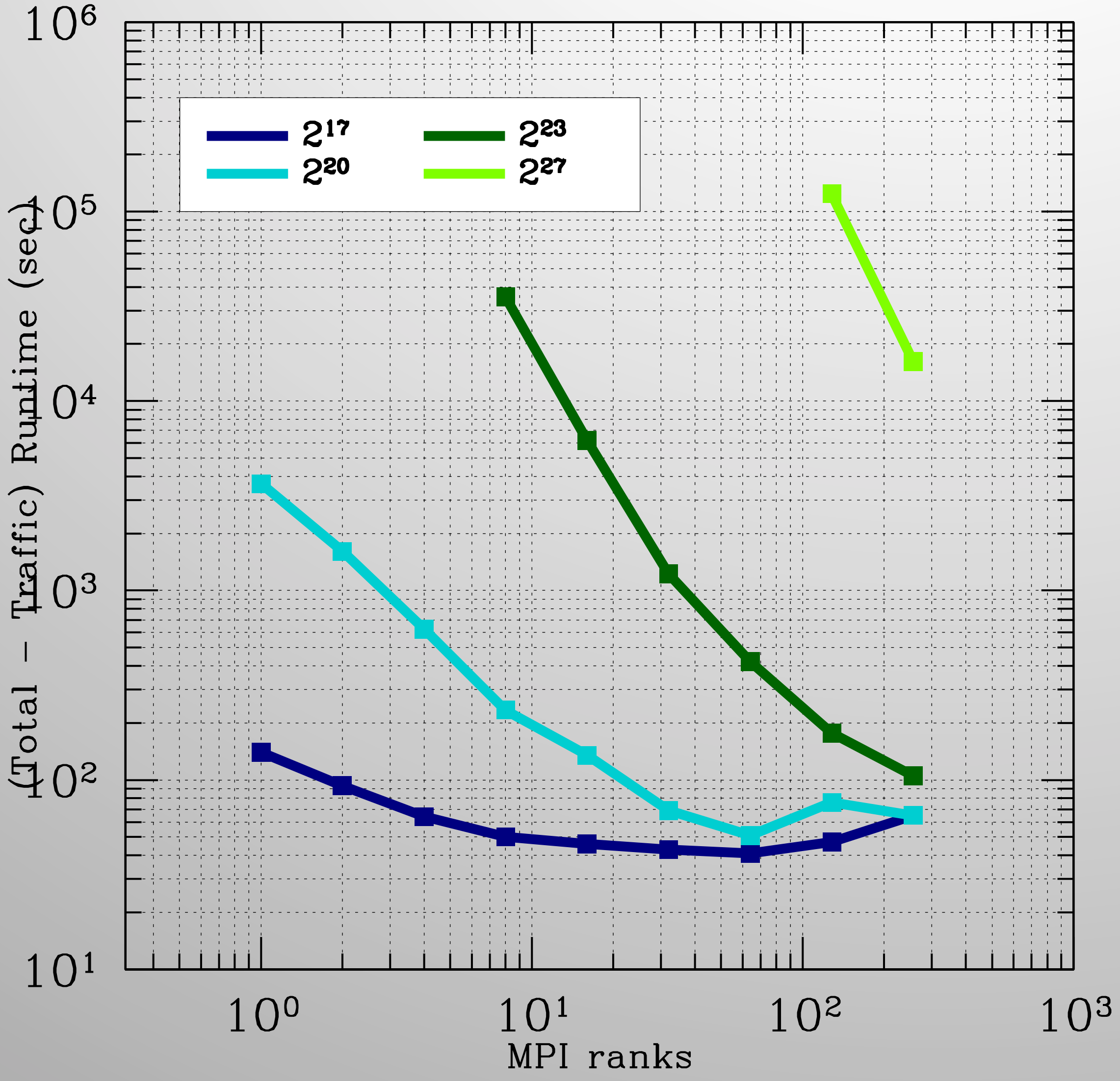


Packet throughput rate

- Packet Rx rate is measured in the steady-state regime (100-200 sim sec)
- NULL scheduler performs worse than YAWNS, and does not scale up as well for this model type, due to densely interconnected rank graph
- The turnover is due to increasing communications overhead
- Larger networks are less efficient for small number of ranks, but become more efficient with increasing number of ranks due to higher compute load which is masking communications



Traffic vs. total runtime

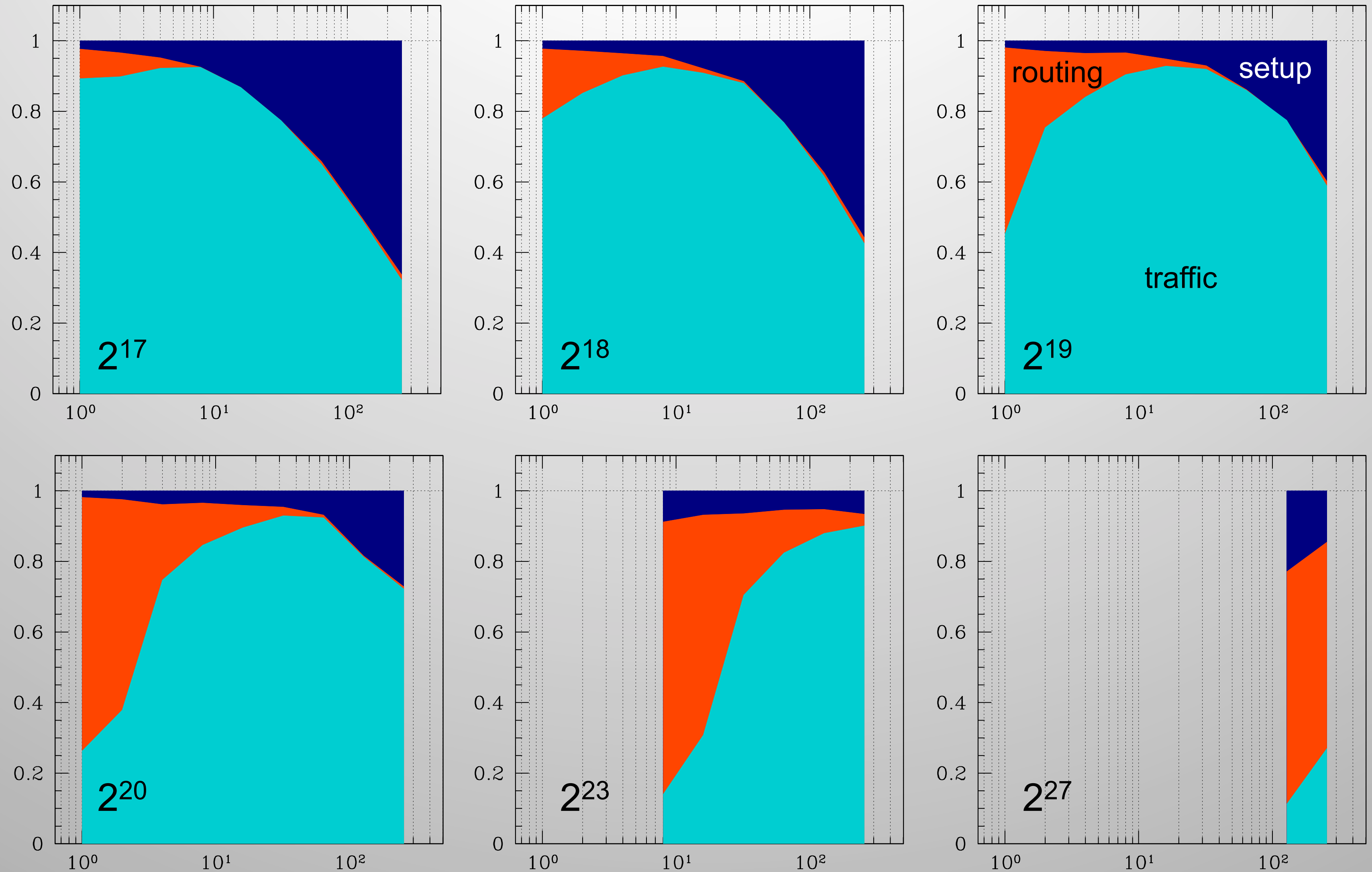


Traffic time refers to packet traffic time, the interval between the first and the last packet arrival. It includes routing calculation time.

Total time is the total runtime, including ns-3 compile, XNDL parsing, model initialization, and routing calculations.

The difference (shown) indicates the impact of the model parsing and initialization.

Execution time fractions

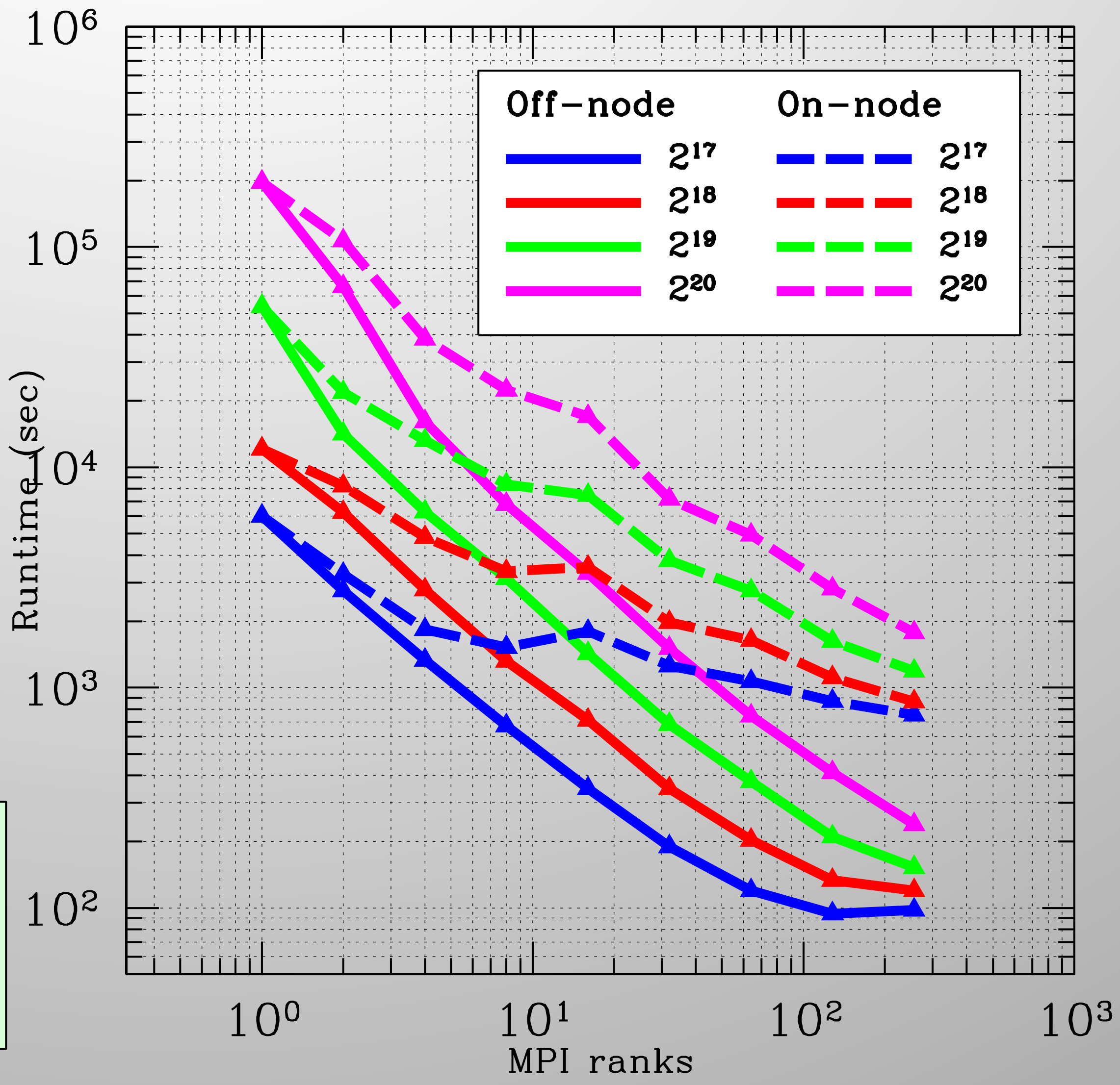


Routing calculations dominate for large model densities

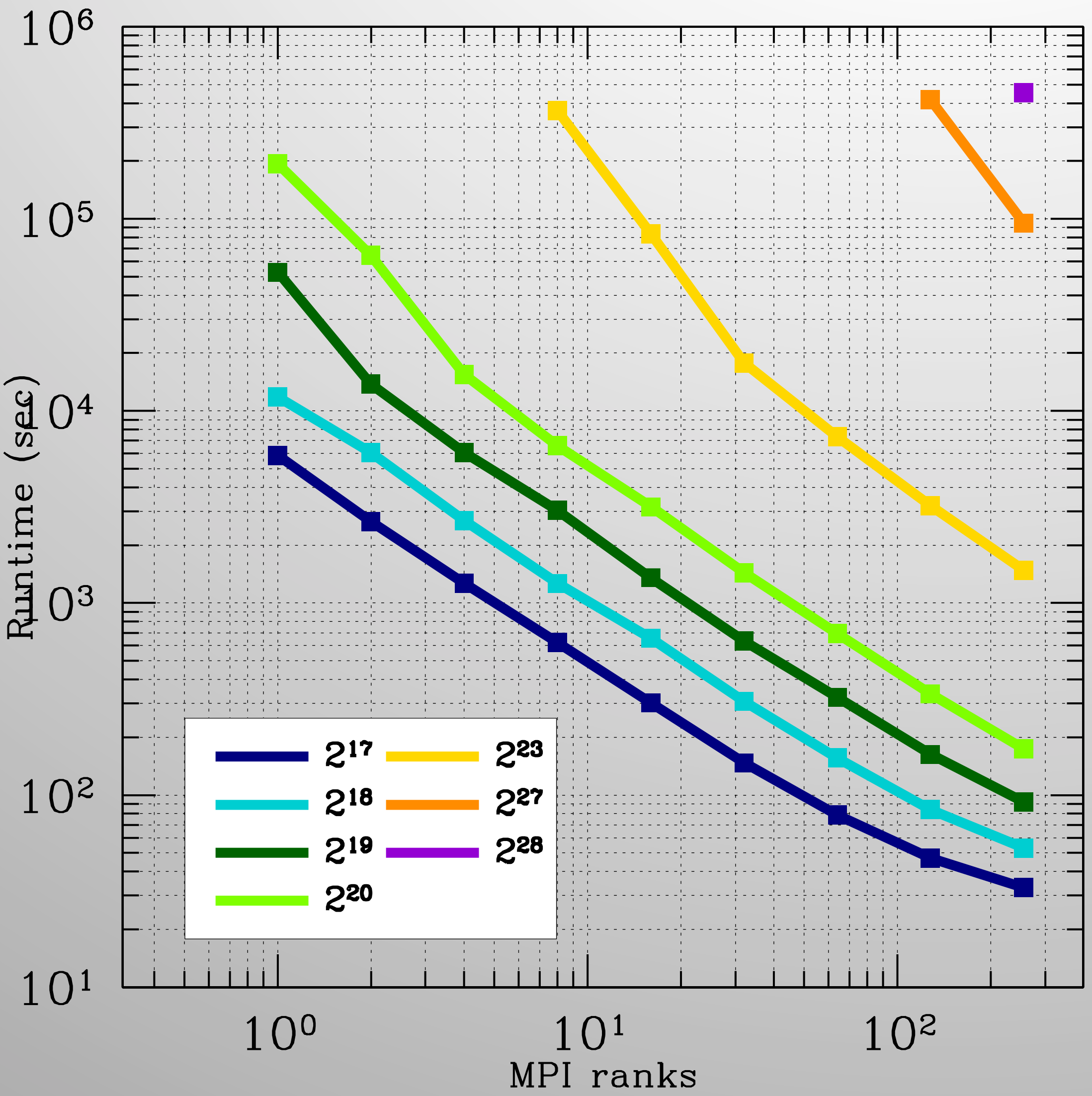
On-node vs. Off-node scaling

- On-node: run on varying number of cores on the same node, up to the node limit
- Off-node: run on varying number of nodes, at 1 rank per node
- Expected runtime metric is inversely proportional to the number of MPI ranks
- The on-node runs are negatively impacted, most likely by node memory bandwidth saturation

Off-node runs are faster, yet are "wasteful", at one rank per node. *On-node* runs are slower, yet let us use more ranks overall.

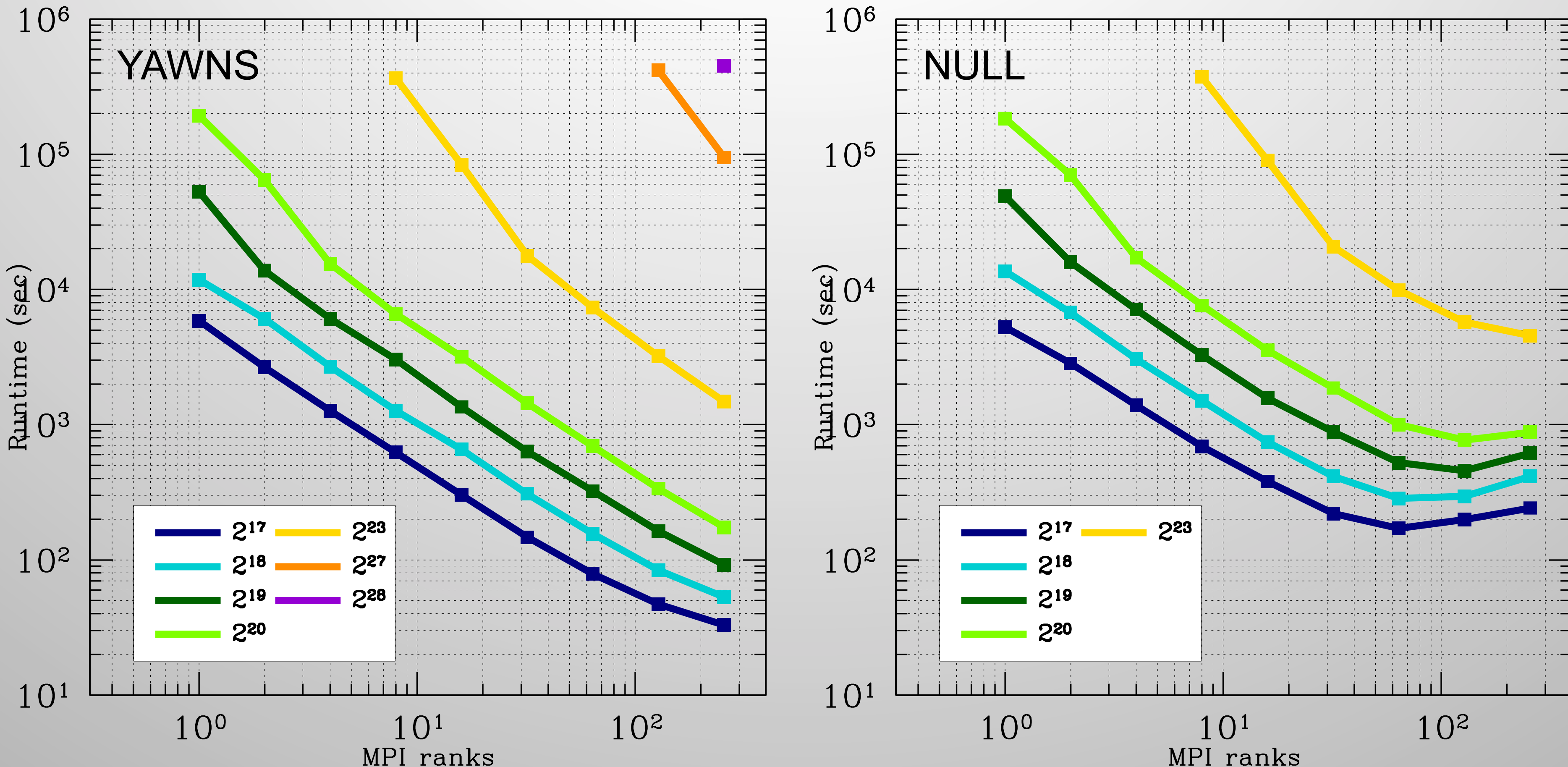


Strong scaling



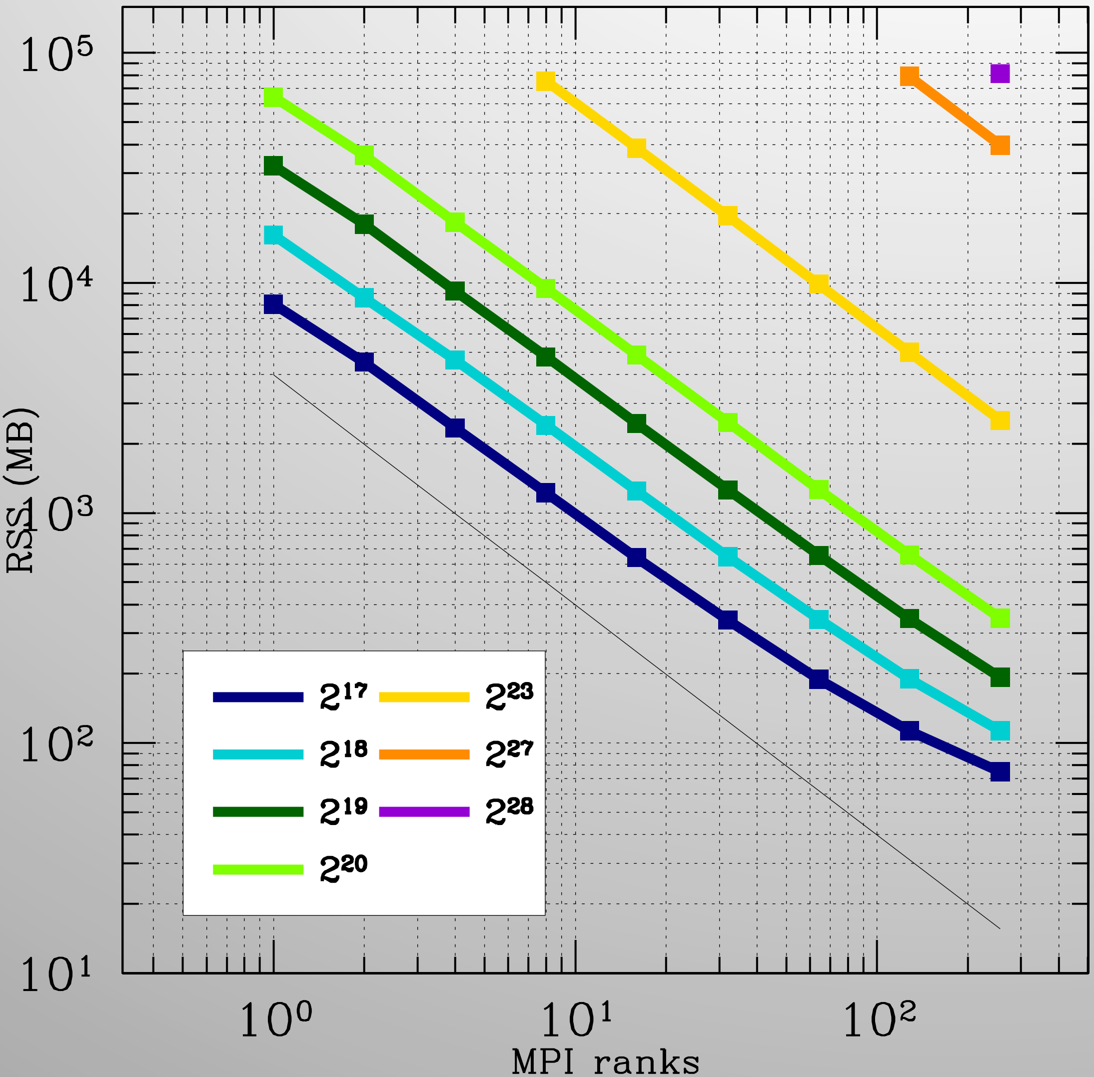
- Fix total workload, increase the number of MPI ranks
- Desired behavior is inversely proportional to the number of ranks
- The slope change indicates the impact of the routing calculations
- At high number of ranks/low model density, the runtime starts to turn over due to increased burden of interprocess communications

YAWNS vs. NULL scheduler



For this type of (densely connected) model graph, NULL-based scheduler cannot take advantage of pair-wise communications, so it performs consistently worse

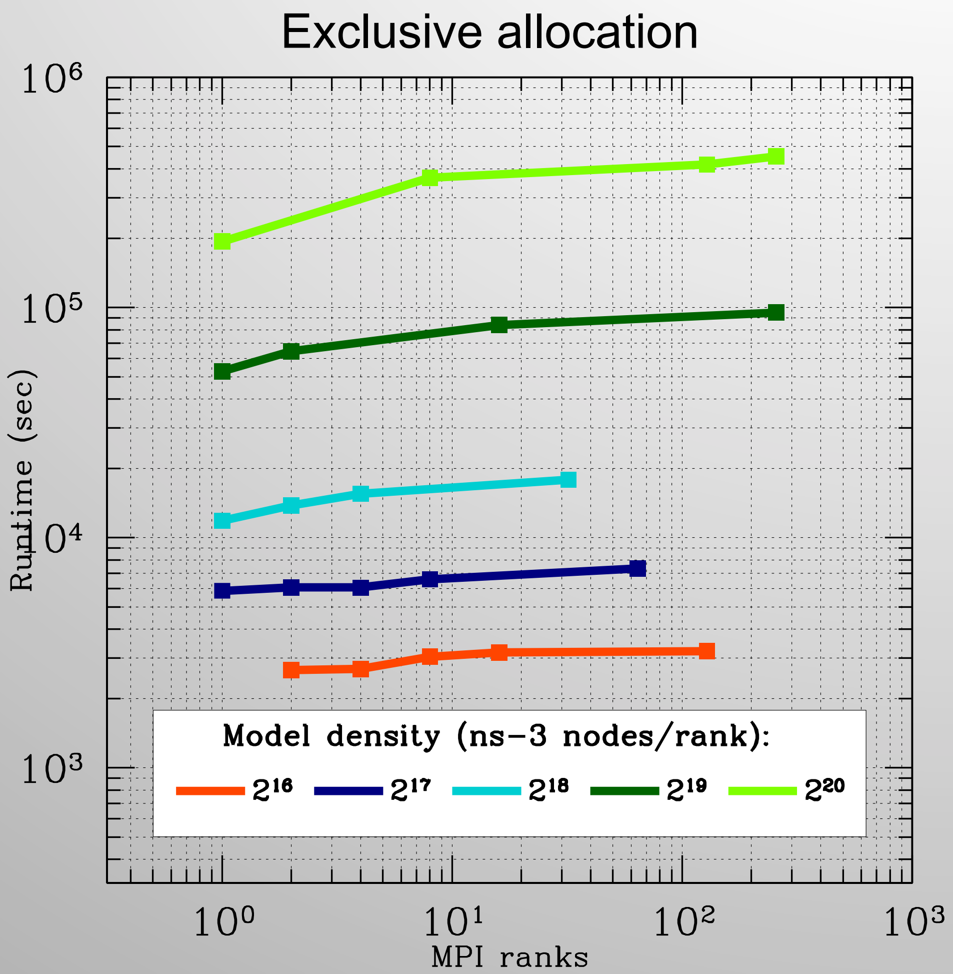
Memory Scaling

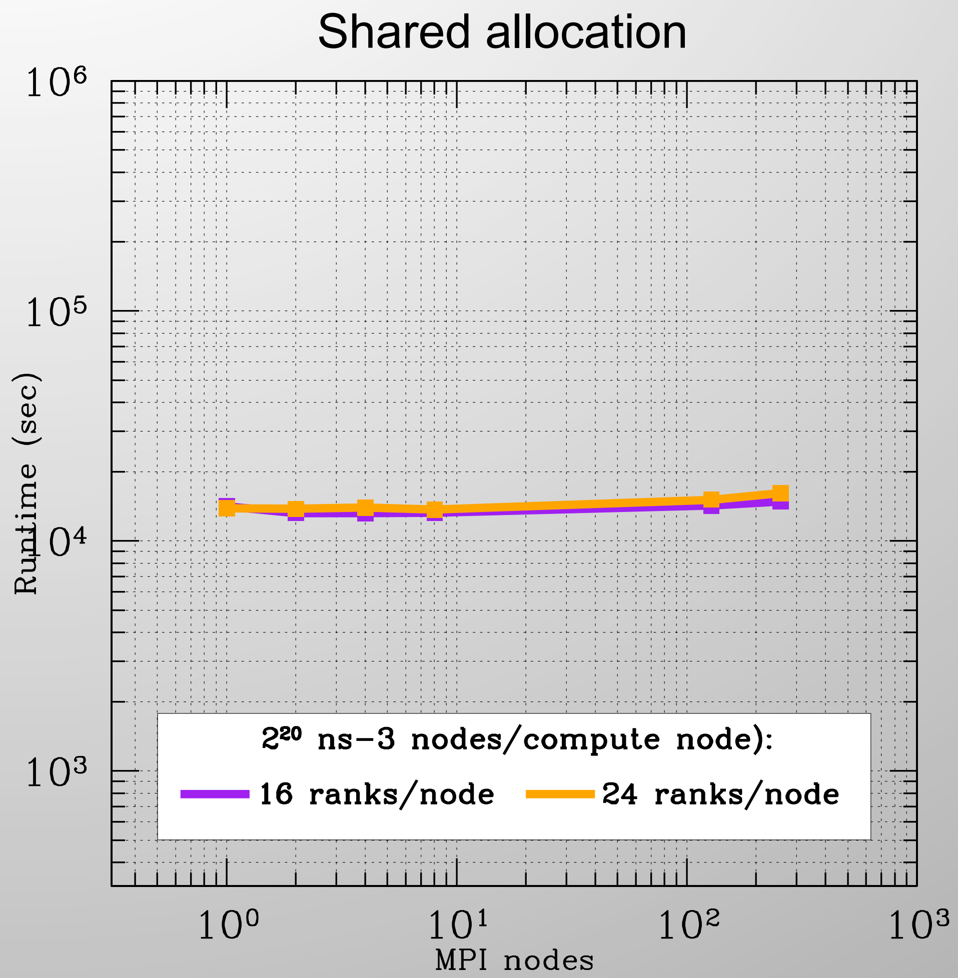


RSS scaling is much more linear than runtime

Hint of flattening is due to fixed size of the code (constant term in RSS)

Weak scaling





In weak scaling, the load per rank is kept constant. The expected behavior is flat (workload increase is balanced by the increase in the number of MPI ranks)

The Quest for 1B Node Model

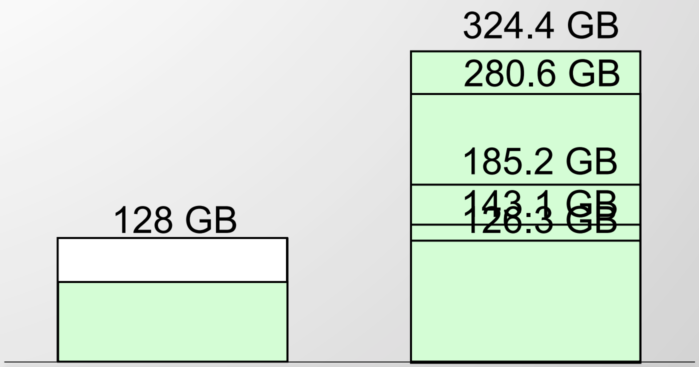
As we were building up models, we realized that 2^{28} model was the largest that could fit into catalyst node (128 GB)

2^{28} model run on 256 nodes at 81.1 GB per node

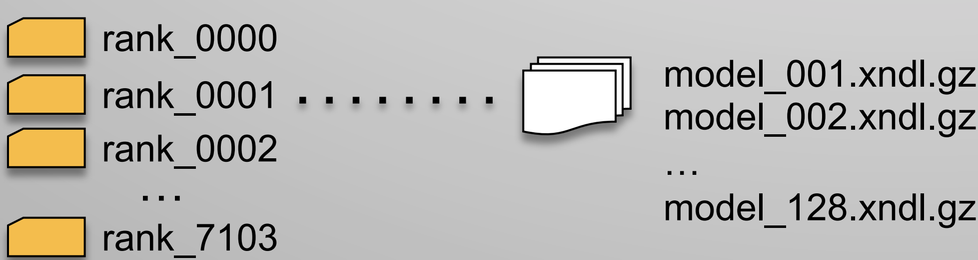
2^{30} model on 256 nodes would then require 324.4 GB

Thus began a quest to optimize RAM for 1B node model:

- Catalyst has 296 nodes: 324.4*(256/296) = 280.6 GB
- Streaming XNDL parser (~34%): 185.2 GB
- UDP packets instead of TCP (~15%): 143.1 GB
- Use 1 PacketSink per node, instead of 2 per channel (6%): 126.3 GB



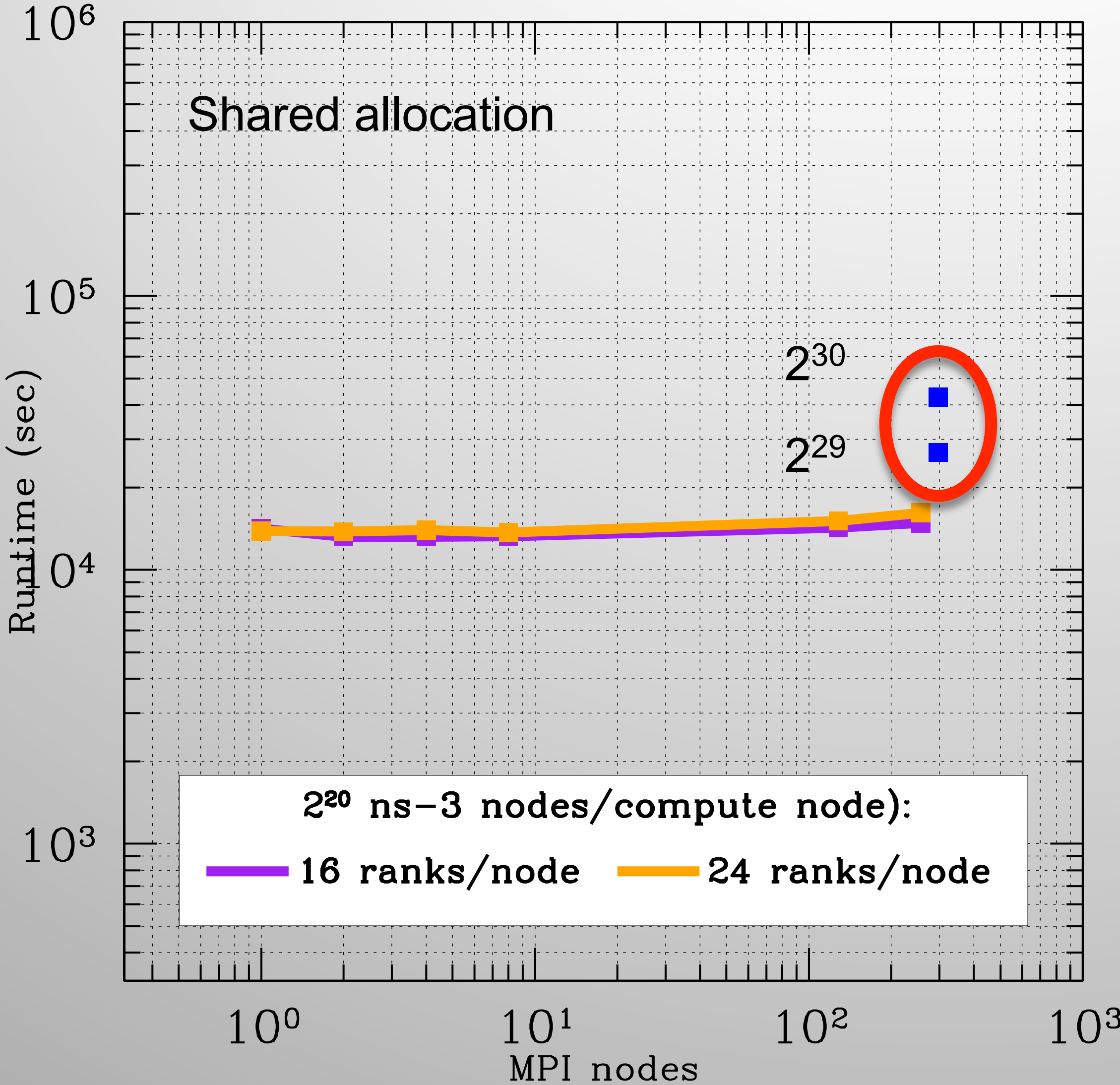
These optimizations allowed us to run the model with 2^{30} (1,073,741,824) ns-3 nodes



2^{30} Model	
Input file (compressed)	Size on disk (GB)
model (orig)	138
sector	21
partition	15
model (split)	400

The final input deck for 2^{30} model consists of 7104 directories (one per rank), with each directory using ~33 MB, for the total size of 400 GB

2^{29} completes; 2^{30} crashes at ~80 sim sec



Error message

internet/model/ipv4-l3-protocol.cc, around line 688:

```
  // 4) packet is not broadcast, and is passed in with a route entry but route->GetGateway is not set (e.g., on-demand)
  if (route && route->GetGateway () == Ipv4Address ())
    {
      // This could arise because the synchronous RouteOutput() call
      // returned to the transport protocol with a source address but
      // there was no next hop available yet (since a route may need
      // to be queried).
      NS_FATAL_ERROR ("Ipv4L3Protocol::Send case 4: This case not yet implemented");
    }
```

Conclusions and Future Work

- New developments in ns-3 world (distributed topology, XNDL, etc.) enable one to build and run large network models, out-of-the-box
- Using moderately large cluster (296 nodes; 7776 cores; 128GB/node) we demonstrated a planetary-scale network model (2^{30} ns-3 nodes)
- Scaling studies (weak and strong) suggest ns-3 scales reasonably well to thousands of ranks
- Distributed scheduler should be chosen to match the network model
- Various optimizations to ns-3 implemented during the course of this work

- Future work will develop XNDL to implement parameter substitution and inheritance; as well as using XSLT to enable sharing models between simulators
- Release the code

Lawrence Livermore National Laboratory