Distributed Simulation with NS-3

Ken Renard
US Army Research Lab

Outline

- Introduction and Motivation for Distributed NS-3
- Parallel Discrete Event Simulation
- MPI Concepts
- Distributed NS-3 Scheduler
- Limitations
- Example Code Walk-through
- Error Conditions
- Performance Considerations
- Advanced Topics

Introduction to Distributed NS-3

- Distributed NS-3 is a scheduler that allows discrete events to be executed concurrently among multiple CPU cores
  - Load and memory distribution
- Initially released in version 3.8
- Implemented by George Riley and Josh Pelkey (Georgia Tech)
- Roots from:
  - Parallel/Distributed ns (pdns)
  - Georgia Tech Network Simulator (GTNetS)
- Performance Studies
  - "Performance of Distributed ns-3 Network Simulator", S. Nikolaev, P. Barnes, Jr., J. Brase, T. Canales, D. Jefferson, S. Smith, R. Soltz, P. Scheibel, SimuTools '13
  - "A Performance and Scalability Evaluation of the NS-3 Distributed Scheduler", K. Renard, C. Peri, J. Clarke , SimuTools '12
    - 360 Million Nodes

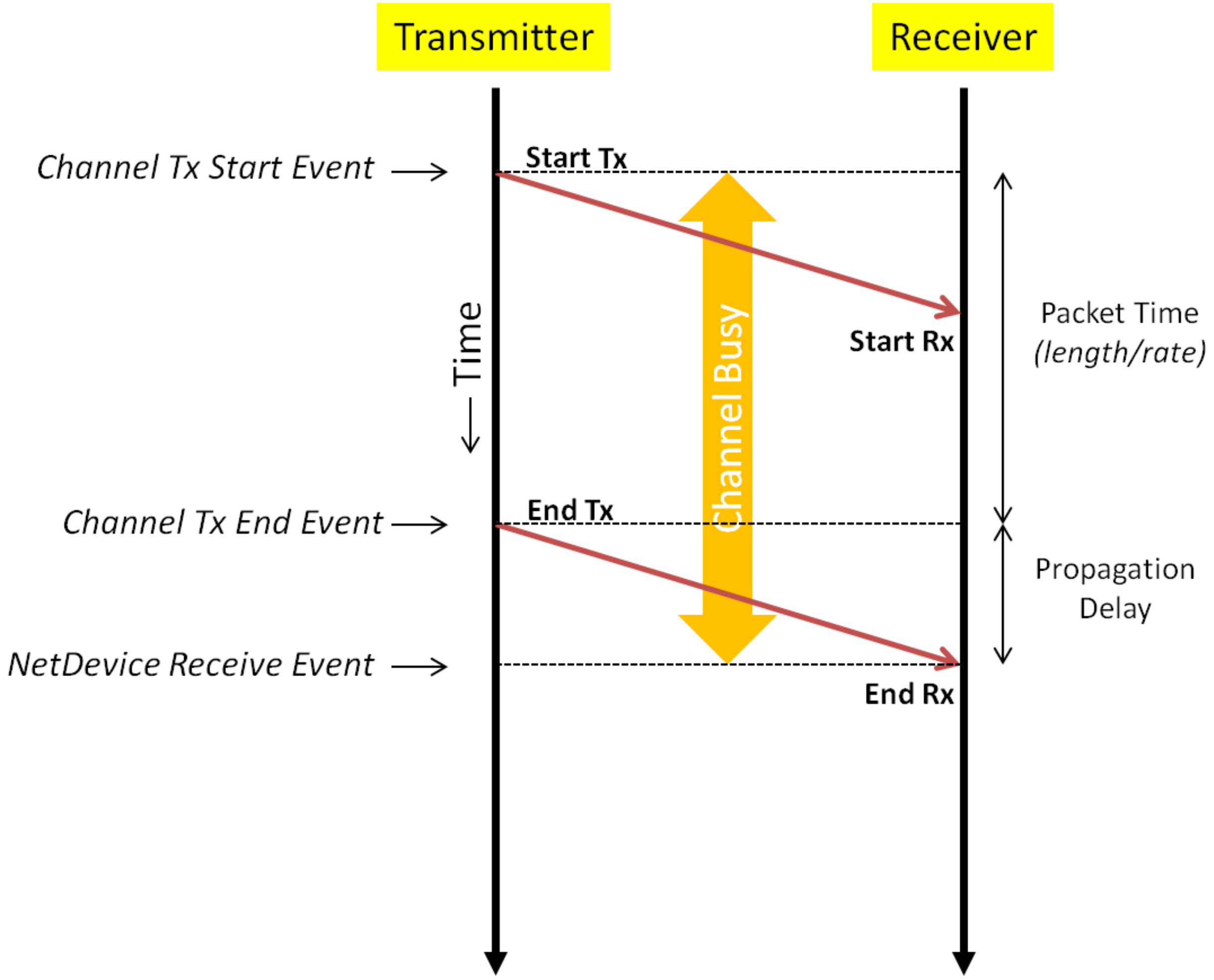
Motivation for High Performance, Scalable Network Simulation

- Reduce simulation run-time for large, complex network simulations
  - Complex models require more CPU cycles and memory
    - MANETs, robust radio devices
    - More realistic application-layer models and traffic loading
    - Load balancing among CPUs
  - Potential to enable real-time performance for NS-3 emulation
- Enable larger simulated networks
  - Distribute memory footprint to reduce swap usage
  - Potential to reduce impact of N^2 problems such as global routing
- Allows network researchers to run multiple simulations and collect significant data

Discrete Event Simulation

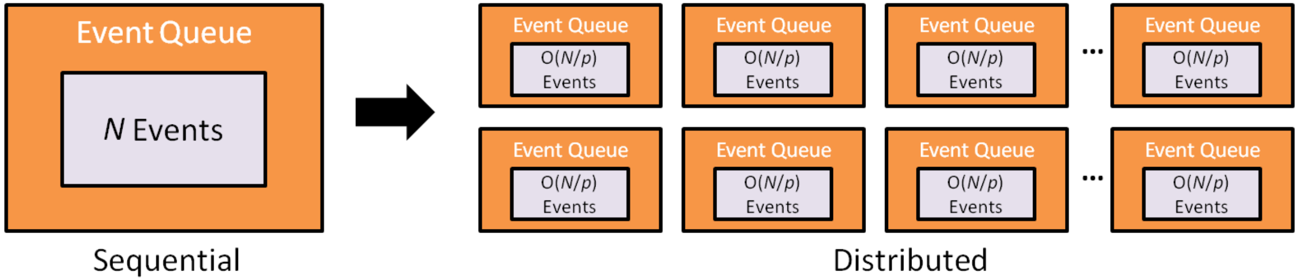
- Execution of a series of time-ordered events
  - Events can change the state of the model
  - Create zero or more future events
- Simulation time advances based on when the next event occurs
  - Instantaneously skip over time periods with no activity
  - Time effectively stops during the processing of an event
- Events are executed in time order
  - New events can be scheduled "now" or in the future
  - New events cannot be scheduled "in the past"
  - Events that are scheduled at the exact same time may be executed in any order
- To model a process that takes time to complete, schedule a series of events that happen at relative time offsets
  - Start sending packet: set medium busy, schedule stop event
  - Stop sending packet: set medium available, schedule receive events
- Exit when there are no more events are in the queue

Discrete Events and Timing for a Packet Transmission



Parallel Discrete Event Simulation (Conservative)

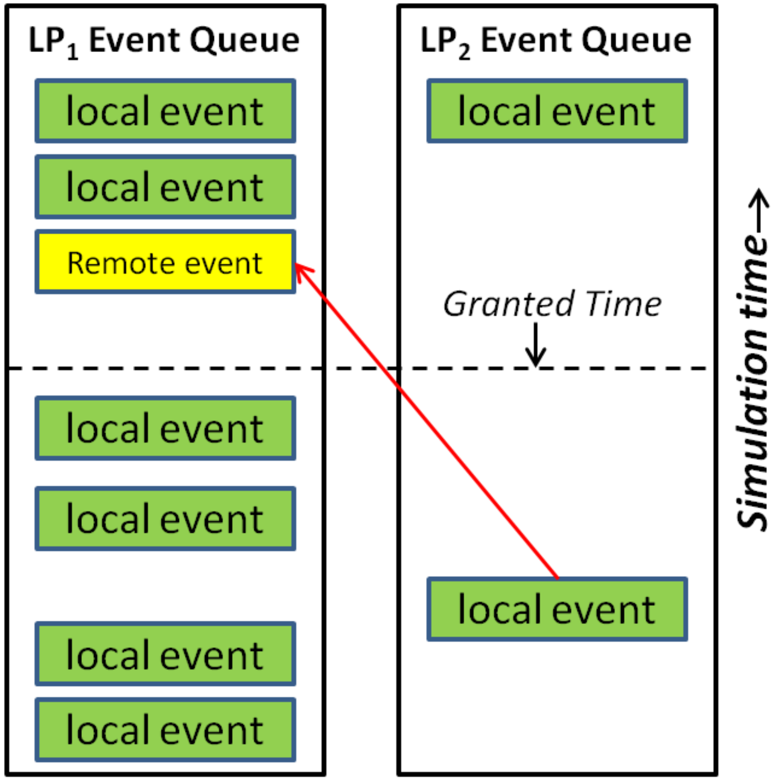
- By partitioning the model (network) into multiple pieces and map these pieces to *Logical Processes*, *(LPs)*, each LP has its own set of events to process
  - LPs are synchronized copies of NS3 running at the same time



- Try to distribute event load (processing load) equally among LPs
  - Exploit parallelism in simulation
- At some point, we will need to schedule an event that will be executed on another LP
  - Messages are passed between LPs to communicate event details and scheduling information
  - Some form of time synchronization is required between LPs
  - Must maintain causality – cannot schedule an event "in the past"
  - We need to communicate our event to a remote LP before that LP's simulation time passes our event time
- Events across LPs can execute independently and in parallel

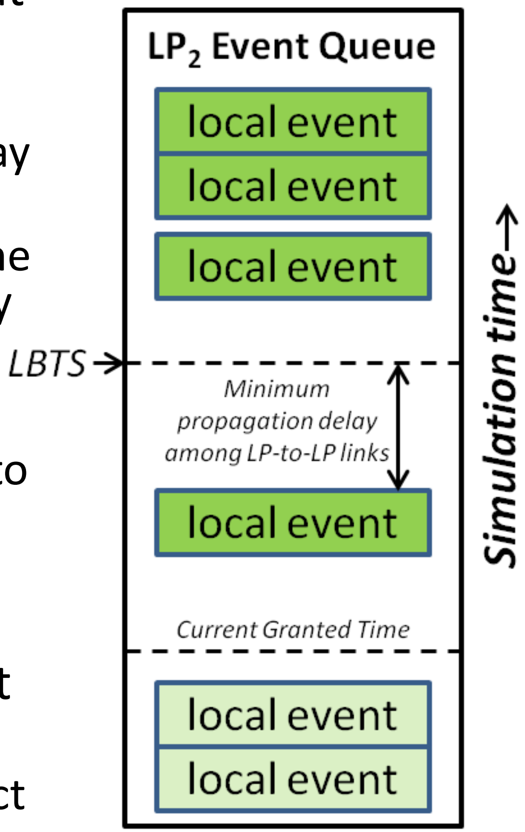
Clock Synchronization in Conservative PDES

- We grant each LP a future time value such that no incoming events will occur before that time
  - In the simple case, all LPs are granted the same time
  - All LPs advance time in synchronized "chunks"
- The LP can now execute all events up to that time while preserving causality
  - Incoming event requests are queued
    - Incoming events will occur after the granted time
- The LP waits until it is granted additional time
  - Even distribution of workload limits wasted time
- We want to maximize grant time such that a larger set of events can be computed in parallel



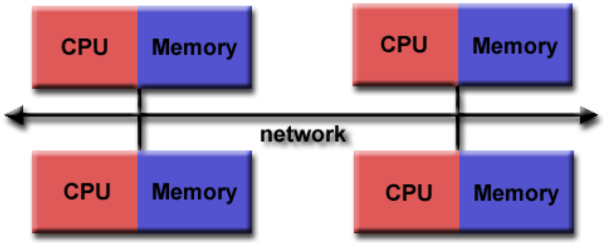
Lookahead & Grant Time Computation

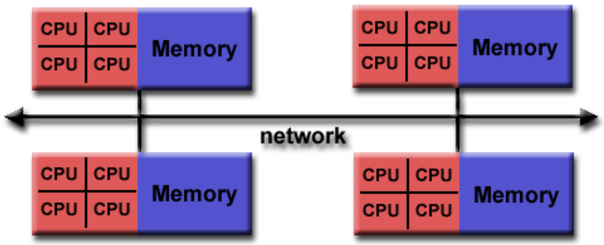
- **Lookahead** value is the minimum amount of time that must elapse before an event at an LP can effect *anything* in another LP
  - In network simulation we can use the propagation delay over a link/channel as the basis for lookahead
  - Among a set of LPs, the maximum lookahead is the time of the next event, plus the minimum propagation delay among links that span LPs
- Compute Lower Bound Time Step (LBTS)
  - Smallest timestamp of an event that can be delivered to another LP
  - Select lowest LBTS over all LPs as global grant time
    - All LPs advance to the same grant time before repeating
- Getting *all* LPs to communicate and determine lowest LBTS can be expensive
  - O(n) to O(n^2) messages, interconnect type, interconnect speed



Message Passing Interface (MPI)

- Distributed NS-3 uses MPI for communication and synchronization
- Message Passing Specification (not the library itself)
  - Point-to-Point as well as collective communications
  - Designed for high performance and scalability
  - De-facto standard for distributed computing
- Allows communication between sets of processes (*ranks*)
  - `mpirun -np 10 ./main`
- Language Independent (C, C++, FORTRAN, Java, Python, etc)
- Targeted distributed memory systems, but works nicely on shared memory as well





- Libraries are built to take advantage of underlying hardware
  - Such as drivers for high-speed interconnects
  - Low latency, high throughput
- Implementations: OpenMPI, MPICH, mpi4py, mpiJava, etc

Images: https://computing.llnl.gov/tutorials/mpi/

MPI Concepts

- Communicators
  - A “channel” among a group of processes (unsigned int)
  - Each process in the group is assigned an ID or *rank*
    - Rank numbers are contiguous unsigned integers starting with 0
    - Used for directing messages or to assign functionality to specific processes
      - `if (rank == 0) print “Hello World”`
  - Default [“everybody”] communicator is MPI_COMM_WORLD
- Point-To-Point Communications
  - A message targeting a single specific process
  - **`MPI_Send(data, data_length, data_type, destination, tag, communicator)`**
    - Data/Data Length – Message contents
    - Data Type – MPI-defined data types
    - Destination – Rank Number
    - Tag – Arbitrary message tag for applications to use
    - Communicator – Specific group where destination exists
  - MPI_Send() / MPI_Isend() – blocking and non-blocking sends
    - MPI_Recv() / MPI_Irevc() – blocking and non-blocking receive

MPI Concepts

- Collective Communications
  - Synchronization – Block until all members of communicator have reached that point
  - Data messaging – Broadcast, scatter/gather, all-to-all
  - Collective Computation – One rank collects data from all ranks and performs an operation (sum, avg, min, max)
- Data Types – select examples
  - MPI_CHAR, MPI_UNSIGNED_CHAR
  - MPI_SHORT, MPI_LONG, MPI_INT
  - MPI_FLOAT, MPI_DOUBLE, MPI_COMPLEX
  - Derived types – built from primitives
- Specifying where processes are run
  - Use config file to specify hosts and #CPUs to run on
    - **`--hostfile`** file for OpenMPI
  - Cluster systems usually have queuing system or scheduler interfaces where host/CPU mapping is done

```
# This is an example hostfile.  Comments begin with #
#
# The following node is a single processor machine:
foo.example.com

# The following node is a dual-processor machine:
bar.example.com slots=2

# The following node is a quad-processor machine, and we
# absolutely want to disallow over-subscribing it:
yow.example.com slots=4 max-slots=4
```

```
#!/bin/csh
#PBS -l walltime=01:00:00
#PBS -l select=128:ncpus=8:mpiprocs=8
#PBS -l place=scatter:excl
#PBS -N myjob
#PBS -q standard

mpirun_shim ${PATH}/big_simulation
```

MPI Programming

OpenMPI Example

- MPI Program Structure
  - Include headers
  - Initialize MPI with command-line args
  - Parallel code
    - Send messages, synchronize
  - Finalize
- Use front-end for compiler
  - **`mpicc, mpicxx, mpif77`**
  - Automatically includes appropriate libraries and include directories
- Use **`mpirun`** to execute
  - Use config file to specify hosts and #CPUs to run on
    - **`--hostfile`** file for OpenMPI
  - Cluster systems usually have queuing system/scheduler interfaces where host/CPU mapping is done

```
#include <mpi.h>
#include <unistd.h>      // For getpid()

int
main (int argc, char **argv)
{
  int size, rank, rc;

  rc = MPI_Init (&argc, &argv);
  if (rc != MPI_SUCCESS)
    MPI_Abort(MPI_COMM_WORLD, rc);

  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  printf ("Hello World from rank %d of %d
             (%d)\n", rank, size, getpid ());

  MPI_Finalize();
}
```

```
$ mpicxx -o hello hello.cc
$ mpirun -np 4 ./hello
Hello World from rank 3 of 4 (35986)
Hello World from rank 0 of 4 (35983)
Hello World from rank 1 of 4 (35984)
Hello World from rank 2 of 4 (35985)
```

MPI Messaging Example

```
#include <mpi.h>
int main (int argc, char **argv)
{
  int rank, rc;
  char *msg = (char *)"Hello";
  int msg_len = strlen(msg);
  char in_msg[msg_len + 1];

  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  if (size < 2) {
    printf ("Need more than one rank to communicate\n");
    MPI_Abort(MPI_COMM_WORLD, 0);
  }

  if (rank == 0) {
    int dest = 1;
    rc = MPI_Send (msg, msg_len, MPI_CHAR, dest,
                   0, MPI_COMM_WORLD);
  }

  if (rank == 1) {
    int count = 0;
    MPI_Status stat;

    rc = MPI_Recv (&in_msg, msg_len, MPI_CHAR,
                   MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &stat);
    in_msg[msg_len] = (char) 0;
    MPI_Get_count (&stat, MPI_CHAR, &count);
    printf("Rank %d receive message \"%s\" (%d) from rank
            %d tag %d\n",  rank, in_msg, count,
            stat.MPI_SOURCE, stat.MPI_TAG);
  }

  MPI_Finalize();
}
```

```
$ mpicxx -o send1 send1.cc
$ mpirun -np 4 ./send1
Rank 1 receive message "Hello" (5) from rank 0 tag 0
$
```

MPI Collective Example -- Barrier

```c
#include <mpi.h>
#include <unistd.h>
#include <stdlib.h>

int
main (int argc, char **argv)
{
  int size, rank, rc;

  rc = MPI_Init (&argc, &argv);
  if (rc != MPI_SUCCESS)
    MPI_Abort(MPI_COMM_WORLD, rc);

  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  MPI_Barrier (MPI_COMM_WORLD);

  srand (getpid ());
  int count = rand() % 1000000000;

  int sum = 0;
  for (int i=0; i < count; i++) {
    sum += rand () % 1000000;
  }

  printf("Rank %d: done with spin (%d)\n",
          rank, count);
  MPI_Barrier (MPI_COMM_WORLD);
  printf("Rank %d: Final Barrier\n", rank);

  MPI_Finalize();
}
```

```
$ time mpirun -np 4 ./coll
Rank 0: done with spin (11587458)
Rank 3: done with spin (171572520)
Rank 2: done with spin (402449947)
Rank 2: Final Barrier
Rank 1: done with spin (777659848)
Rank 1: Final Barrier
Rank 3: Final Barrier
Rank 0: Final Barrier

real        0m10.151s
user        0m36.471s
sys         0m0.050s


$ time mpirun -np 4 ./coll
Rank 1: done with spin (30229414)
Rank 0: done with spin (258675938)
Rank 3: done with spin (496367588)
Rank 1: Final Barrier
Rank 2: done with spin (731537290)
Rank 2: Final Barrier
Rank 0: Final Barrier
Rank 3: Final Barrier

real        0m9.621s
user        0m34.365s
sys         0m0.043s
```

MPI Collective Example -- AllGather

```c
#include <mpi.h>
#include <unistd.h>
#include <stdlib.h>

int
main (int argc, char **argv)
{
  int size, rank, rc;

  rc = MPI_Init (&argc, &argv);
  if (rc != MPI_SUCCESS)
    MPI_Abort(MPI_COMM_WORLD, rc);

  MPI_Comm_size (MPI_COMM_WORLD, &size);
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);

  srand (getpid ());
  int allValues[size];
  int myValue = rand() % 1000000000;

  MPI_Allgather (&myValue, 1, MPI_INT,
                 allValues, 1, MPI_INT,
                 MPI_COMM_WORLD);

 printf ("Rank %d: [", rank);
  for (int i = 0; i < size; i++) {
    printf("%d, ", allValues[i]);
  }
  printf ("]\n");

  MPI_Finalize();
}
```

```
$ mpirun -np 4 ./gather
Rank 3: [29003797, 719191937, 424799615, 114846810, ]
Rank 0: [29003797, 719191937, 424799615, 114846810, ]
Rank 1: [29003797, 719191937, 424799615, 114846810, ]
Rank 2: [29003797, 719191937, 424799615, 114846810, ]
```

Distributed NS-3

1. Configuring and Building Distributed NS-3
2. Basic approach to Distributed NS-3 simulation
3. Memory Optimizations
4. Discussion of works-in-progress to simplify and optimize distributed simulations

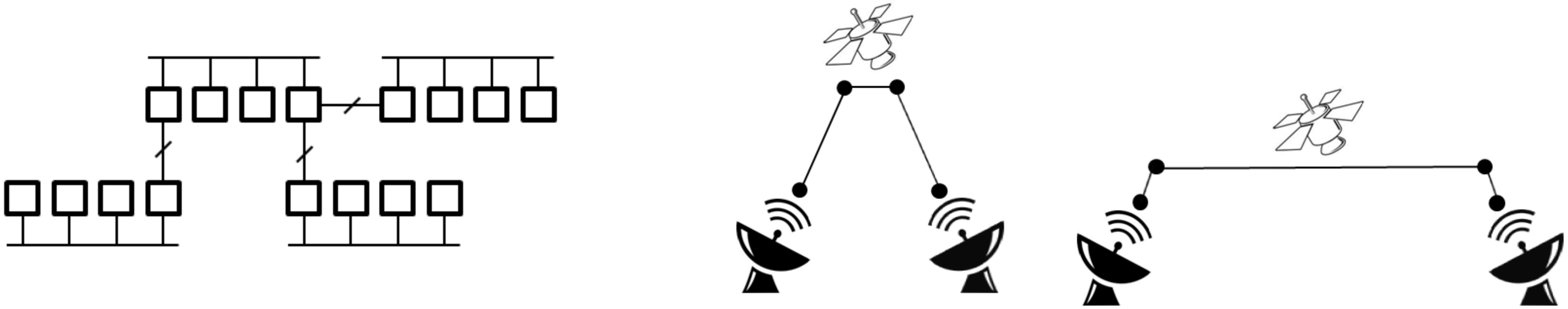
Building Distributed NS-3

- Add “--enable-mpi” to ‘waf configure’ line
  - Tries to run ‘mpic++’
    - Recognizes OpenMPI and MPICH libraries
  - Defines “NS3_MPI” and either “NS3_OPENMPI” or “NS3_MPICH”

```
---- Summary of optional NS-3 features:
Python Bindings               : not enabled (PyBindGen missing)
BRITE Integration             : not enabled (BRITE not enabled (see option --with-brite))
NS-3 Click Integration        : not enabled (nsclick not enabled (see option --with-nsclick))
GtkConfigStore                : enabled
XmlIo                         : enabled
Threading Primitives          : enabled
Real Time Simulator           : enabled
Emulated Net Device           : enabled
File descriptor NetDevice     : enabled
Tap FdNetDevice               : enabled
Emulation FdNetDevice         : enabled
PlanetLab FdNetDevice         : not enabled (PlanetLab operating system not detected…
Network Simulation Cradle     : not enabled (NSC not found (see option --with-nsc))
MPI Support                   : enabled
NS-3 OpenFlow Integration     : not enabled (OpenFlow not enabled (see option --with-openflow))
SQlite stats data output      : enabled
```

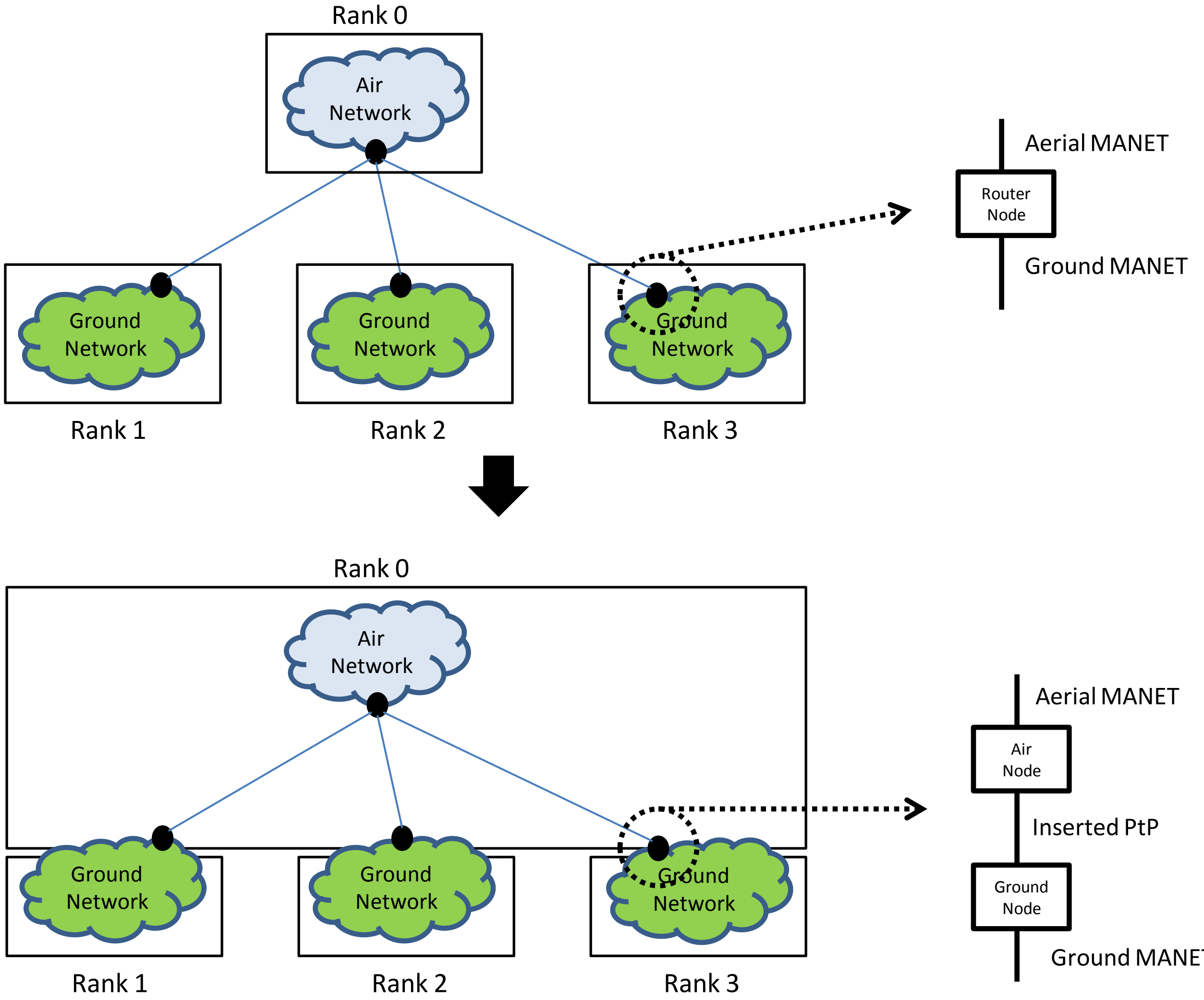
Building a Distributed NS-3 Simulation

- Choose partitioning strategy
  - Find obvious sections of the network that will operate most independently
    - Minimize communication between partitions
  - Find large latencies in network
    - Large latencies are large (good) lookahead values
- Build topology as normal, assigning “SystemId” values on all Nodes
  - **`CreateObject<Node> (rankId)`**
- Distributed NS-3 can only be partitioned over Point-to-Point (P2P) links
  - A special type of P2P will be created by the PTPHelper if Nodes do not have the same systemId [PointToPointRemoteChannel]
  - P2P links can be “inserted” where latency is available
  - Latency can sometimes be “moved” around



Distributed NS-3 Partitioning Example

- Set of ground MANETs
- Aerial layer MANET
- Cannot partition at wireless link between ground and air networks
    - Place node from air network in each ground network
    - Insert Point-to-Point link
- Possibly "move" latency from air-to-ground into PtP link

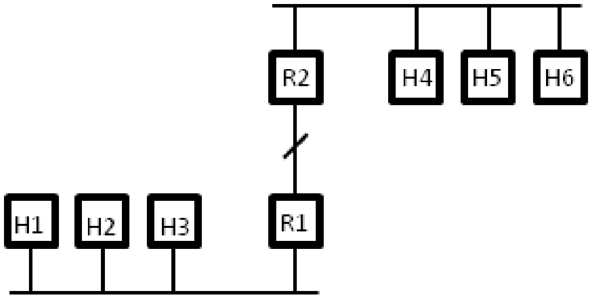


Distributed NS-3
Load Distribution

- All ranks create all nodes and links
    - Setup time and memory requirements are similar to sequential simulation
    - Event execution happens in parallel
    - Memory is used for nodes/stacks/devices that “belong” to other ranks
- Non-local nodes do not have to be fully configured
    - Application models should not be installed on non-local nodes
    - Stacks and addresses probably should be installed on non-local nodes
        - So that global routing model can ‘see’ the entire network
- When packets are transmitted over P2P-Remote links, the receive event is communicated to the receiving rank
    - Send event immediately, do not wait for grant time
    - Receive event is added to remote rank’s queue instead of local
- At end of grant time
    - Read and schedule all incoming events
    - Compute and negotiate next grant time

Sending a Packet to Remote Rank

- Consider 2 CSMA networks connected by a single P2P link
  - One router on each network that spans P2P and CSMA networks
  - A packet is sent from H1 to H6 via R1 and R2
  - At R1, packet is forwarded on to P2P link R1<->R2
- When Packet is sent to P2P-Remote Channel
  - Instead of scheduling a receive on the destination PTPDevice, we call **`MpiInterface::SendPacket()`**
- **`MpiInterface::SendPacket()`**
  - Arguments
    - Packet data
    - Receive time – Packet time plus link delay
    - Remote SystemId (rank)
    - Remote nodeId
    - Remote InterfaceId
  - Serializes packet and destination data
  - MPI_Isend() byte stream to remote rank



RxTime	NodeId	DeviceId	Packet Data

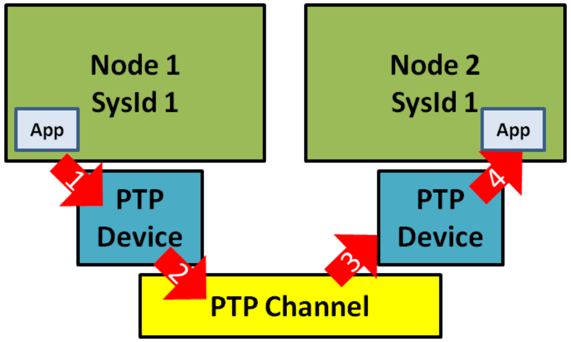
Serialization of packet transmit event over PTP-Remote Channel in Distributed NS-3

Receiving a Packet from Remote Rank

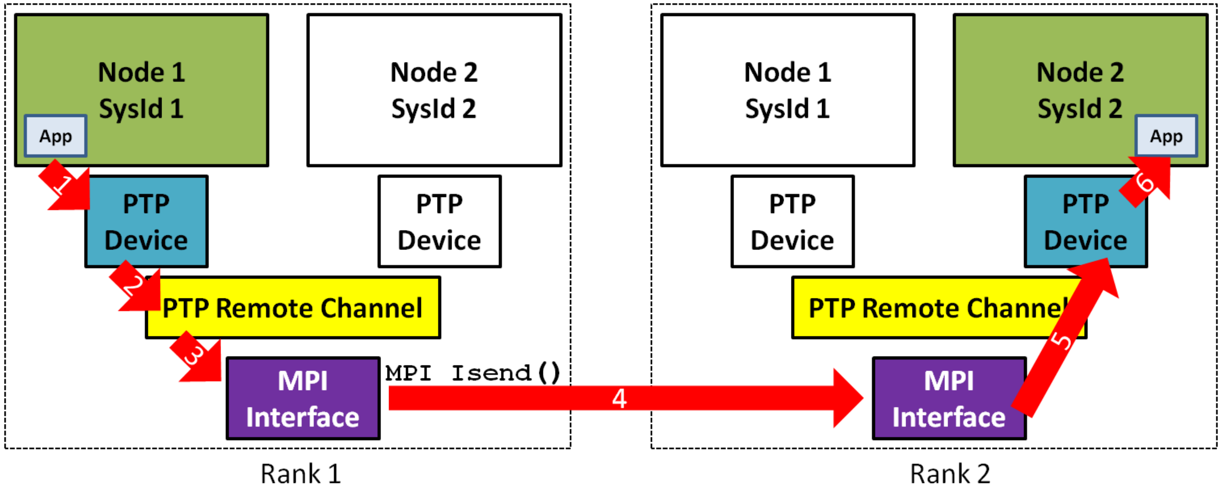
- At granted time, read all MPI message from wire
- For each message
  - Deserialize target ***Receive Time***, ***Node*** and ***InterfaceId***
  - Deserialize packet
  - Find Node by ID
  - Find NetDevice on node with correct interfaceId
  - Get **`MpiReceiver`** object which is aggregated to the NetDevice
    - MpiReceiver is a small shim that passes receive events to the proper NetDevice callback
  - Schedule Receive event @RxTime
    - **`MpiReceiver::Receive()`**
      - This calls its callback which set is to PointToPointNetDevice::Receive() by the PointToPoint helper.

Sending a Packet to a Remote Rank

Sequential

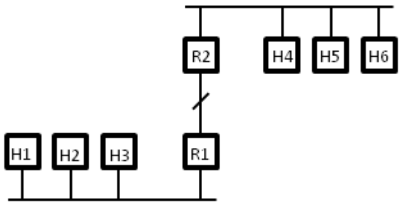


Distributed

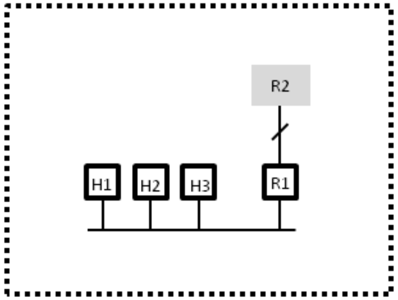


Distributed NS-3
Load and Memory Distribution

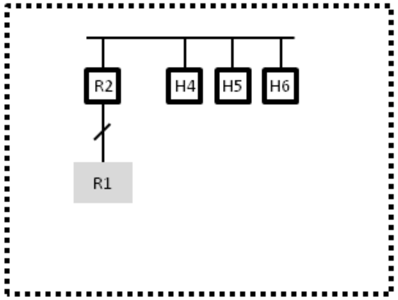
- Save memory by not creating nodes/stacks/links that “belong” in other LPs
  - Exception is “ghost” nodes that bridge LP borders
    - Ghost node creation is only necessary as a convenience
- Requires *manual intervention*
  - Global and NIX routing do not see entire topology
    - Add static, default routes manually
    - Hint: IPv6 allows for more “aggregatable” routes
  - Node indexing is not symmetric
    - If R1 or R2 have different node numbers in each LP, then **`MpiInterface::SendPacket()`** will select the wrong destination
  - Interface identifiers must align in same fashion



Sequential

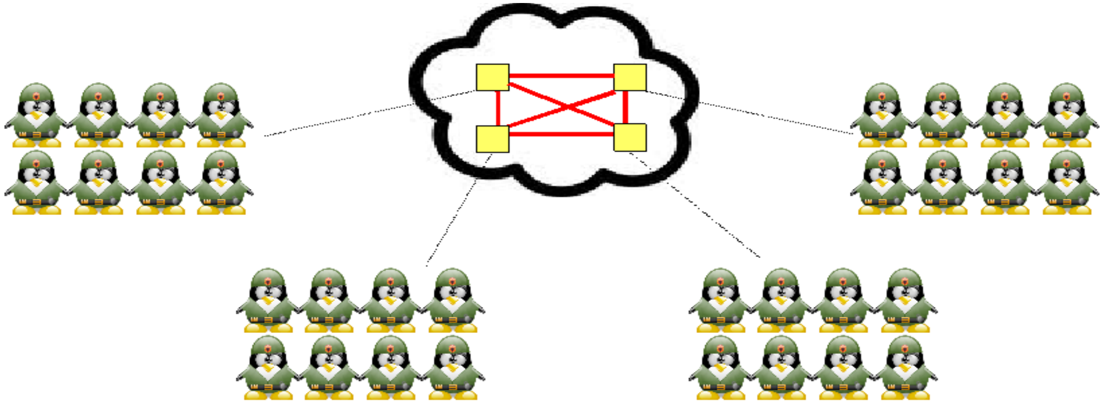


Rank 1



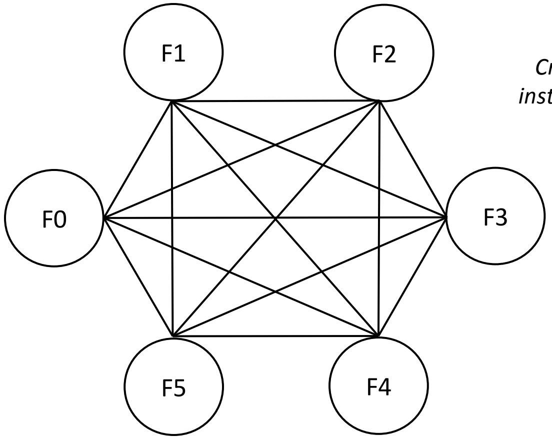
Rank 2

Node and Interface “Alignment”



- “Router-in-the-sky” scenario
- N^2 mesh of interconnected nodes at central hub

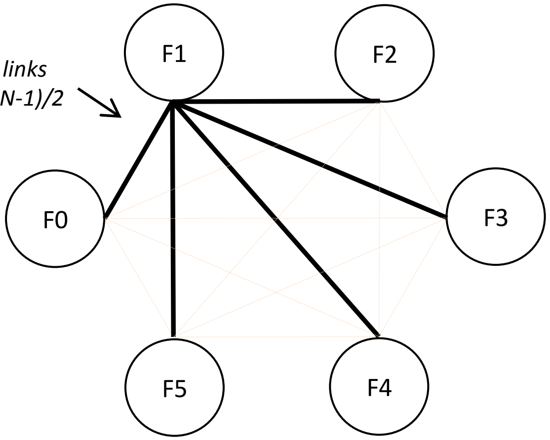
Inter-Federate “Mesh”



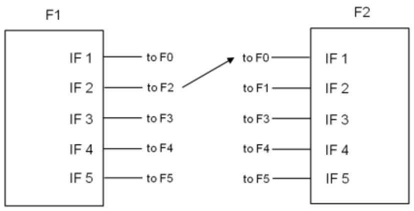
Inter-Federate “Mesh”

Federate 1 perspective

Create (N-1) links instead of N(N-1)/2*



Packets from F1 go to 1st interface on remote Federates



Limitations of Distributed NS3

- Partitioning is a manual process
- Partitioning is restricted to Point-To-Point links only
  - Partitioning within a wireless network is not supported
    - Lookahead is very small and dynamic
- Need full topology in all LPs
  - Exception with careful node ordering, interface numbering, and manual routing

Example Code

src/mpi/examples/third-distributed.cc

Need to include mpi.h →

```
#ifdef NS3_MPI
#include <mpi.h>
#endif

// Default Network Topology (same as third.cc from tutorial)
// Distributed simulation, split along the p2p link
// Number of wifi or csma nodes can be increased up to 250
//
//   Wifi 10.1.3.0
//                 AP
//  *    *    *    *
//  |    |    |    |    10.1.1.0
// n5   n6   n7   n0 -------------- n1   n2   n3   n4
//                   point-to-point  |    |    |    |
//                                   ================
//                          |          LAN 10.1.2.0
//                          |
//                 Rank 0   |   Rank 1
// -------------------------|-----------------------------

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("ThirdExampleDistributed");
```

Example Code

src/mpi/examples/third-distributed.cc

```
                  int
                  main (int argc, char *argv[])
                  {
                  #ifdef NS3_MPI
                    // Distributed simulation setup
   Enable MPI →     MpiInterface::Enable (&argc, &argv);
Set Scheduler →     GlobalValue::Bind ("SimulatorImplementationType",
                                       StringValue ("ns3::DistributedSimulatorImpl"));

  Rank Number →     uint32_t systemId = MpiInterface::GetSystemId ();
         Size →     uint32_t systemCount = MpiInterface::GetSize ();

                    // Check for valid distributed parameters.
                    // Must have 2 and only 2 Logical Processors (LPs)
   Size Check →     if (systemCount != 2)
                      {
                        std::cout << "This simulation requires 2 and only 2 logical
                                      processors.” << std::endl;
                        return 1;
                      }
```

[Command line parsing and LogEnable]

Example Code

src/mpi/examples/third-distributed.cc

Node Rank 0 →

Node Rank 1 →

Nothing different here →

CSMA net node on Rank 1 →

```
NodeContainer p2pNodes;
Ptr<Node> p2pNode1 = CreateObject<Node> (0); // Create node w/ rank 0
Ptr<Node> p2pNode2 = CreateObject<Node> (1); // Create node w/ rank 1
p2pNodes.Add (p2pNode1);
p2pNodes.Add (p2pNode2);

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma, 1);  // Create csma nodes with rank 1

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Example Code

src/mpi/examples/third-distributed.cc

Wifi net on Rank 0 →

```
NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi, 0); // Create wifi nodes with rank 0
NodeContainer wifiApNode = p2pNodes.Get (0);

YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());

WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac“, "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));

NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);

mac.SetType ("ns3::ApWifiMac“, "Ssid", SsidValue (ssid));

NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
```

Example Code

src/mpi/examples/third-distributed.cc

Installing Internet Stacks on everything →

Assigning Addresses to everything →

[Mobility]

```
InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevices);
```

Example Code

src/mpi/examples/third-distributed.cc

Apps for Rank 1 →

```
// If this simulator has system id 1, then
// it should contain the server application,
// since it is on one of the csma nodes
if (systemId == 1)
  {
    UdpEchoServerHelper echoServer (9);
    ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
    serverApps.Start (Seconds (1.0));
    serverApps.Stop (Seconds (10.0));
  }
```

Apps for Rank 0 →

```
// If the simulator has system id 0, then
// it should contain the client application,
// since it is on one of the wifi nodes
if (systemId == 0)
  {
    UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);
    echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
    echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.)));
    echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

    ApplicationContainer clientApps =
              echoClient.Install (wifiStaNodes.Get (nWifi - 1));
    clientApps.Start (Seconds (2.0));
    clientApps.Stop (Seconds (10.0));
  }
```

Example Code

src/mpi/examples/third-distributed.cc

GlobalRouting will work since we have full topology →

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Simulator::Stop (Seconds (10.0));

[Tracing]

Simulator::Run ();
Simulator::Destroy ();
```

Disable MPI →

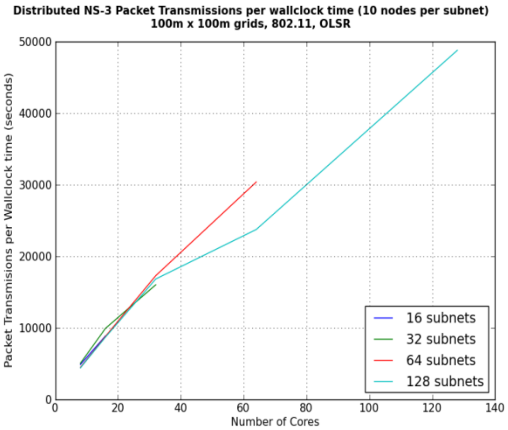
```
// Exit the MPI execution environment
MpiInterface::Disable ();
return 0;
```

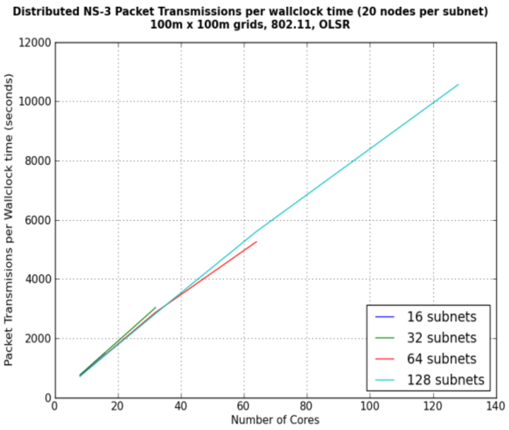
Error Conditions

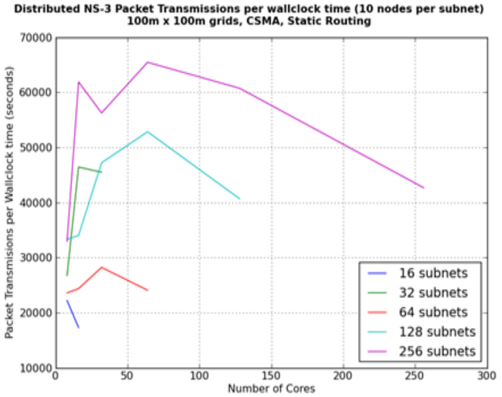
- **`Can't use distributed simulator without MPI compiled in`**
  - Not finding or building with MPI libraries
  - Reconfigure NS-3 and rebuild
- **`assert failed. cond="pNode && pMpiRec", file=../src/mpi/model/mpi-interface.cc, line=413`**
  - Mis-aligned node or interface IDs

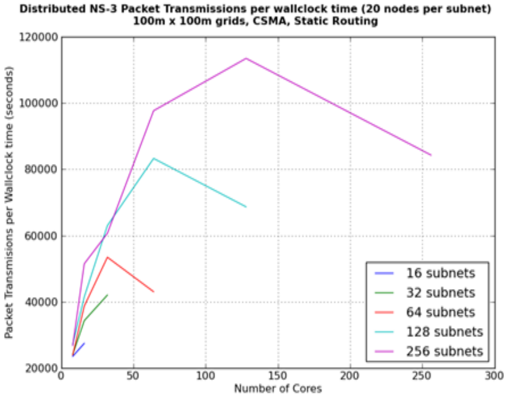
Performance Optimizations

- Memory Optimization
- Larger lookahead (Link latency) helps parallelism
- Cost of the AllGather grows exponentially with LP count
  - If workload per LP is high, fall-off in performance moves to higher LP count
  - With lower workload, performance can fall off at 32-128 LPs
- More work and larger latencies mean better performance of distributed scheduler
- Choose appropriate metric for measuring performance
  - Events/sec can be misleading with varying event cost
  - Packet transmissions (or receives) per wall-clock time


Distributed NS-3 Packet Transmissions per wallclock time (10 nodes per subnet)
100m x 100m grids, 802.11, OLSR


Distributed NS-3 Packet Transmissions per wallclock time (20 nodes per subnet)
100m x 100m grids, 802.11, OLSR


Distributed NS-3 Packet Transmissions per wallclock time (10 nodes per subnet)
100m x 100m grids, CSMA, Static Routing


Distributed NS-3 Packet Transmissions per wallclock time (20 nodes per subnet)
100m x 100m grids, CSMA, Static Routing

Conservative PDES – NULL Message

- An alternative to global synchronization of LBTS
  - Decreases “cost” of time synchronization
- Each event message exchanged includes a new LBTS value from sending LP to receiving LP
  - LBTS is computed for each LP-to-LP message
  - An LP now cares only about its connected set of LPs for grant time calculation
- When there are no event messages exchanged, a “NULL” event message is sent with latest LBTS value
- Advantages to using NULL-message scheduler
  - Less expensive negotiation of time synchronization
  - Allows independent grant times

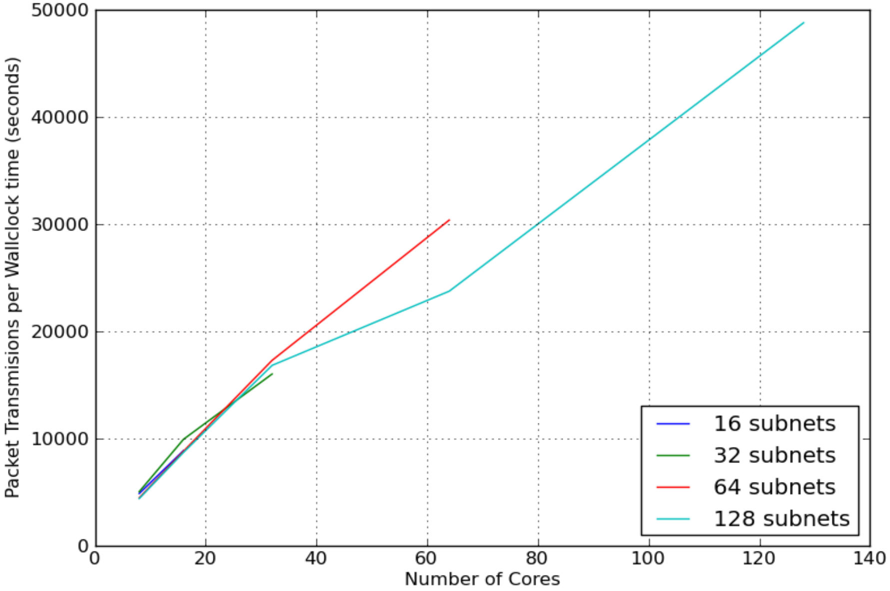
Advanced Topics / Future Work

- Distributed Real Time
  - Versus simultaneous real-time emulations:
    - LP-to-LP messaging can be done with greater lookahead to counter interconnect delay
- Routing
  - AS-like routing between LPs
  - Goal is to enable Global or NIX routing without full topology in each LP
- Alignment
  - Negotiate node and interface IDs at run time
- Partitioning with automated tools
  - Graph partitioning tools
  - Descriptive language to describe results of partitioning to topology generation
- Optimistic PDES
  - Break causality with ability to “roll-back” time
- Partitioning across links other than P2P
- Full, automatic memory scaling
  - Automatic ghost nodes, globally unique node IDs

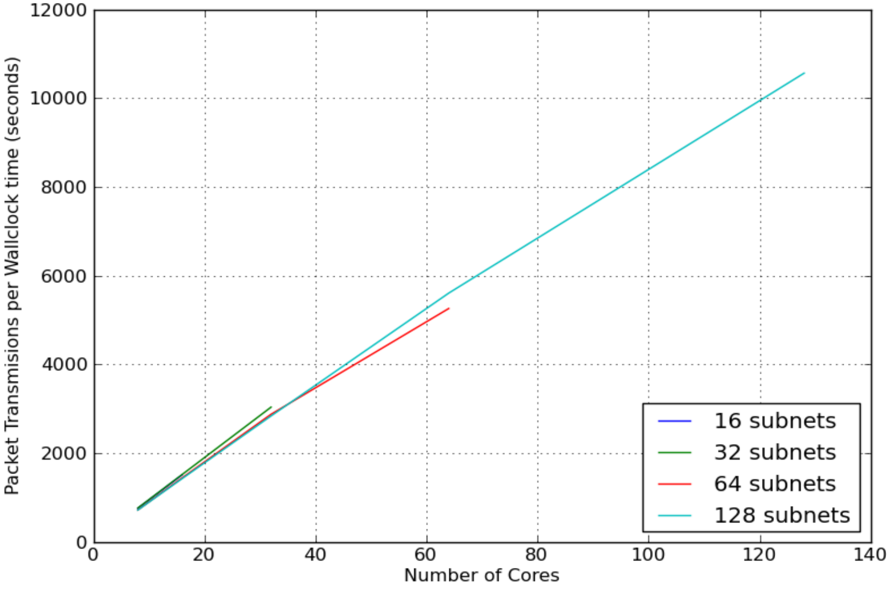
References

- “Parallel and Distributed Simulation Systems”, R. M. Fujimoto, Wiley Interscience, 2000.
- “Distributed Simulation with MPI in *ns-3”,* J. Pelkey, G. Riley, Simutools ‘11.
- “Performance of Distributed ns-3 Network Simulator”, S. Nikolaev, P. Barnes, Jr., J. Brase, T. Canales, D. Jefferson, S. Smith, R. Soltz, P. Scheibel, SimuTools '13.
- “A Performance and Scalability Evaluation of the NS-3 Distributed Scheduler”, K. Renard, C. Peri, J. Clarke , SimuTools '12.

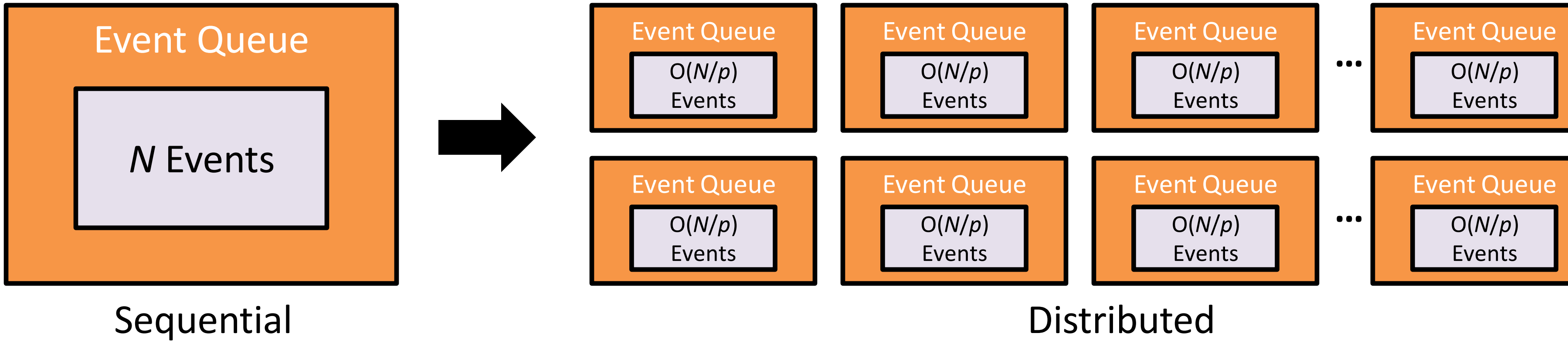
SCRATCH SPACE



Distributed NS-3 Packet Transmissions per wallclock time (10 nodes per subnet)
100m x 100m grids, 802.11, OLSR



Distributed NS-3 Packet Transmissions per wallclock time (20 nodes per subnet)
100m x 100m grids, 802.11, OLSR



Event Queue

N Events

Sequential

Event Queue $O(N/p)$ Events

Event Queue $O(N/p)$ Events

Event Queue $O(N/p)$ Events

…

Event Queue $O(N/p)$ Events

Event Queue $O(N/p)$ Events

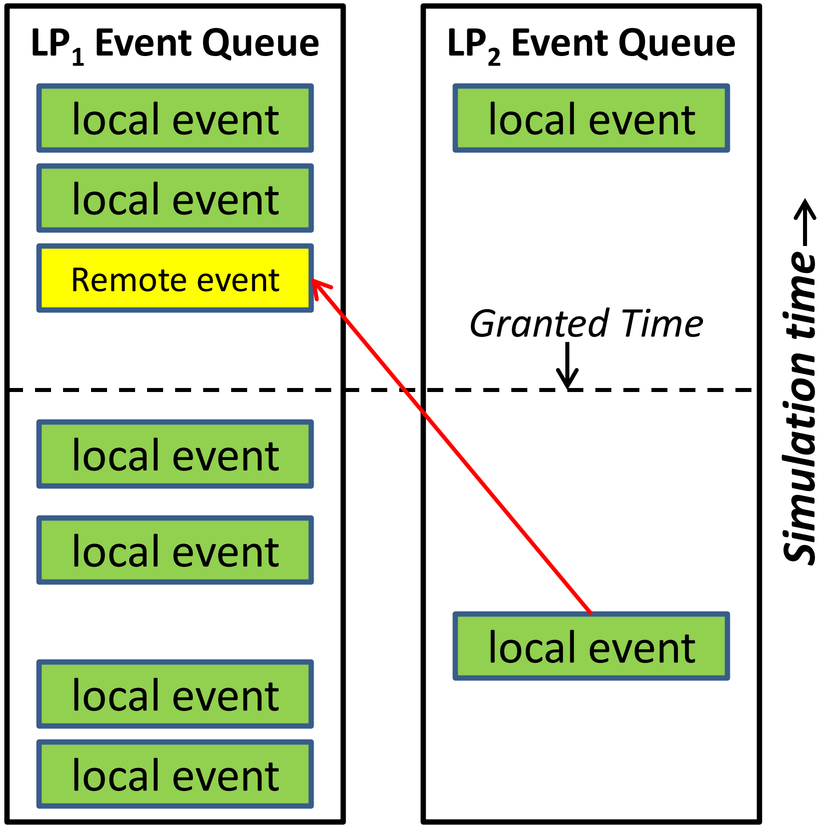
Event Queue $O(N/p)$ Events

Event Queue $O(N/p)$ Events

…

Event Queue $O(N/p)$ Events

Distributed



LP$_1$ Event Queue

local event

local event

Remote event

local event

local event

local event

local event

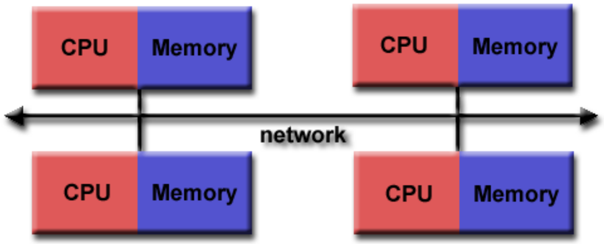
LP$_2$ Event Queue

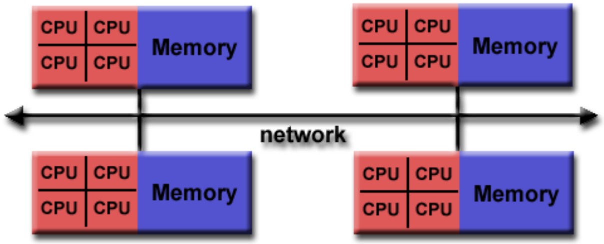
local event

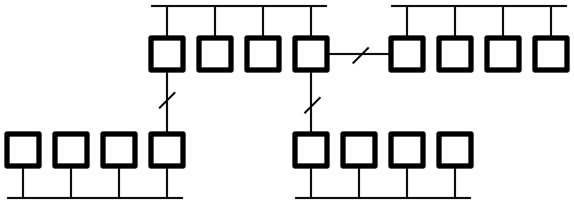
Granted Time

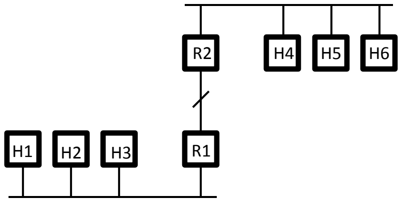
local event

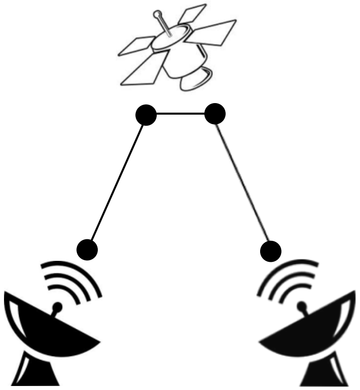
Simulation time →

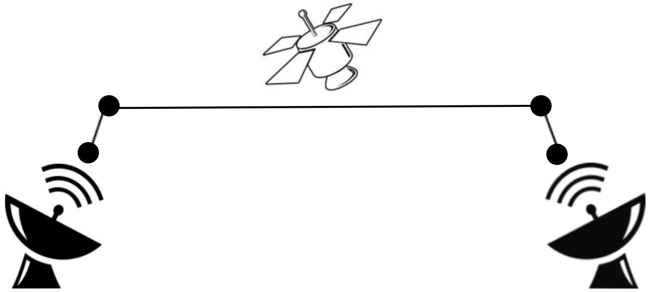


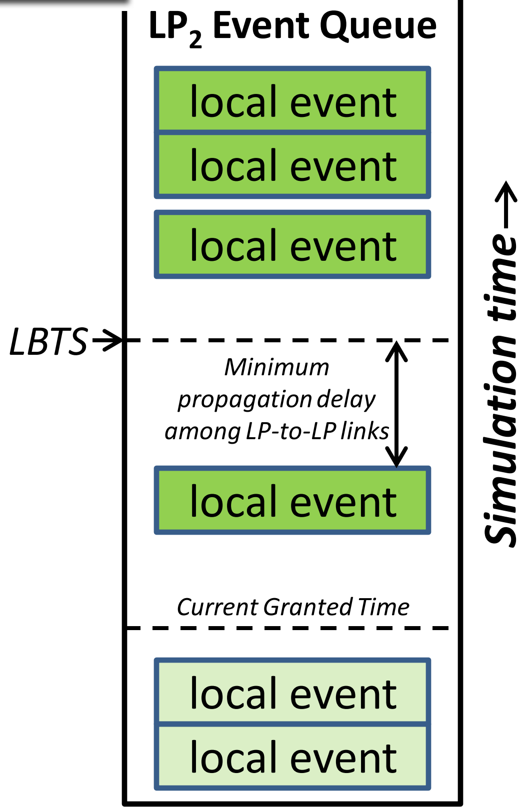












LP$_2$ Event Queue

local event

local event

local event

LBTS

Minimum propagation delay among LP-to-LP links

local event

Current Granted Time

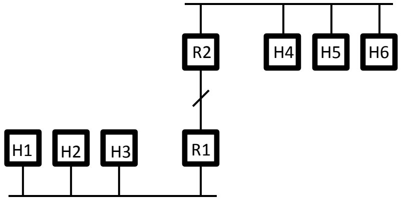
local event

local event

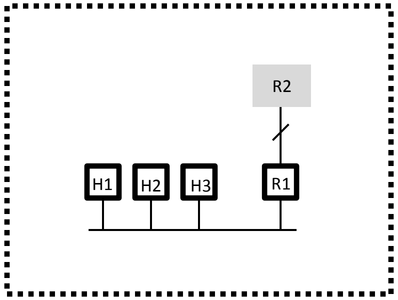
Simulation time

RxTime	NodeId	DeviceId	Packet Data

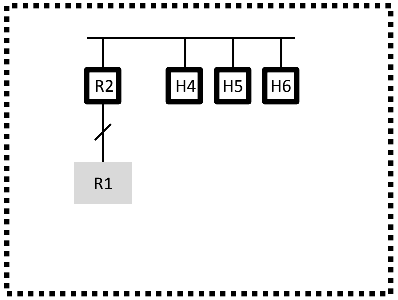
Serialization of packet transmit event over PTP-Remote Channel in Distributed NS-3



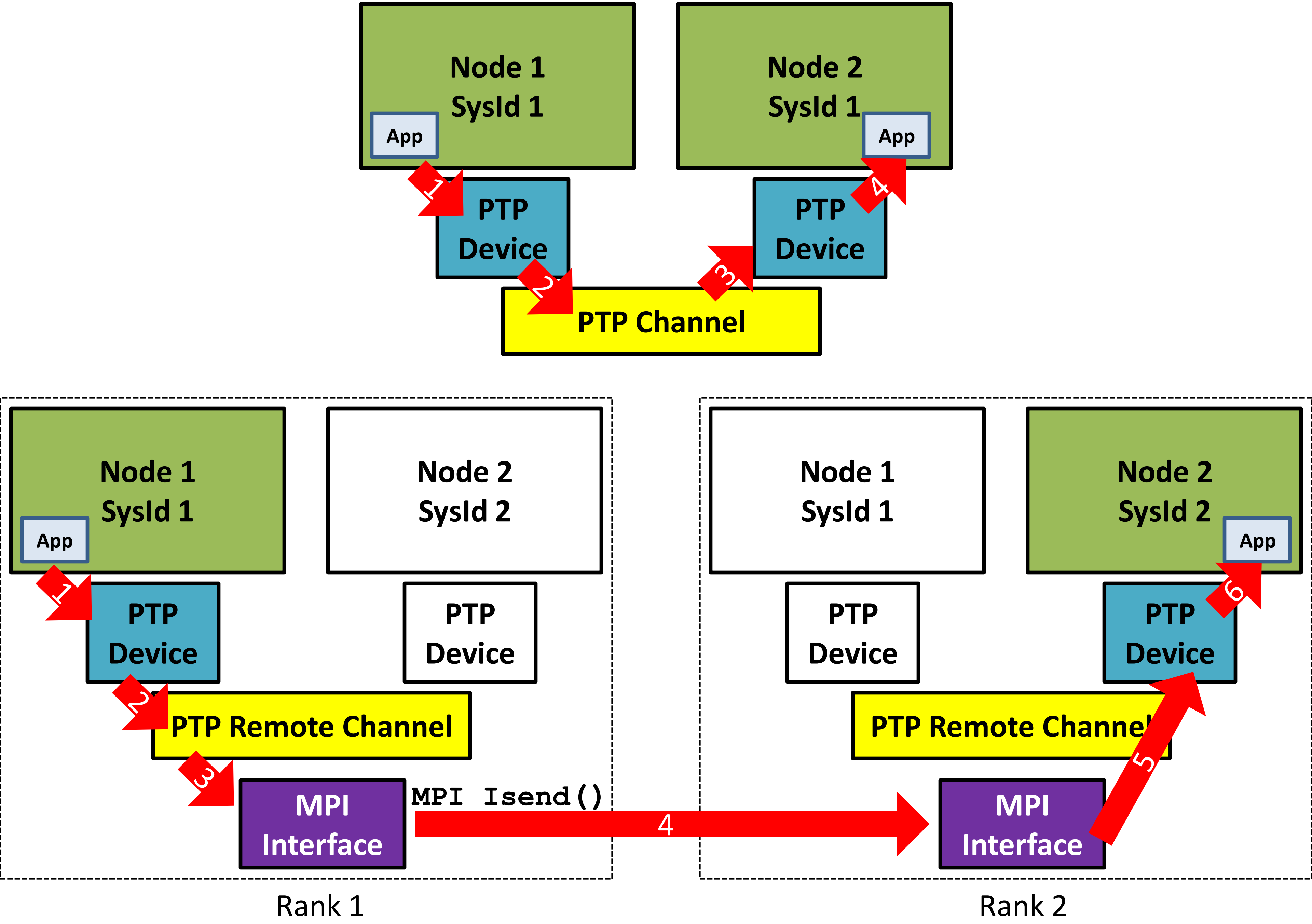
Sequential

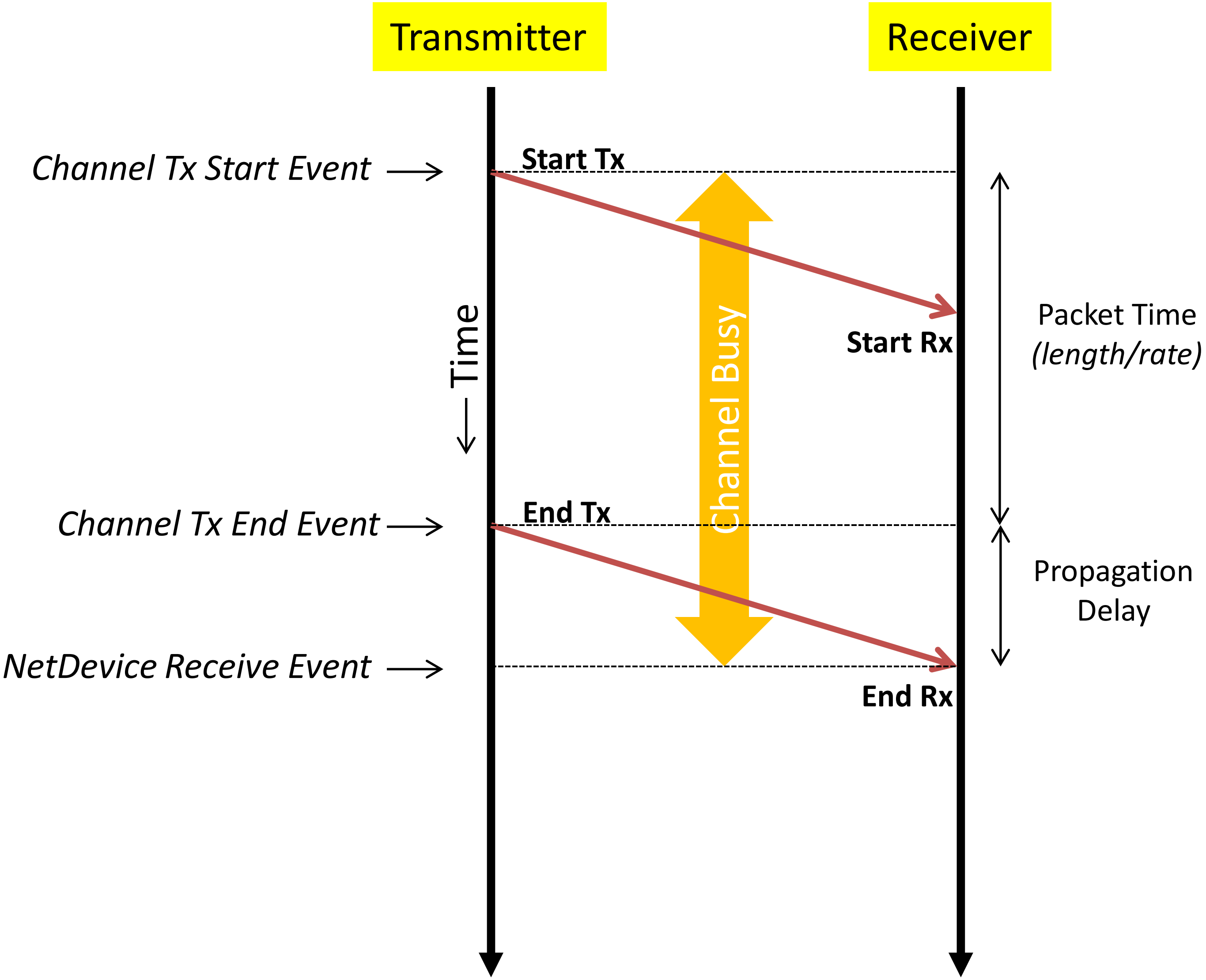


Rank 1



Rank 2


Node 1
SysId 1
App
Node 2
SysId 1
App
1
PTP Device
4
2
3
PTP Device
PTP Channel
Node 1
SysId 1
App
Node 2
SysId 2
1
PTP Device
PTP Device
2
PTP Remote Channel
3
MPI Interface
MPI_Isend()
4
Rank 1
Node 1
SysId 1
Node 2
SysId 2
App
6
PTP Device
PTP Device
PTP Remote Channel
5
MPI Interface
Rank 2



Transmitter

Receiver

Channel Tx Start Event →

Start Tx

Channel Busy

Time ⟶

Start Rx

Packet Time

(length/rate)

Channel Tx End Event →

End Tx

Propagation Delay

NetDevice Receive Event →

End Rx