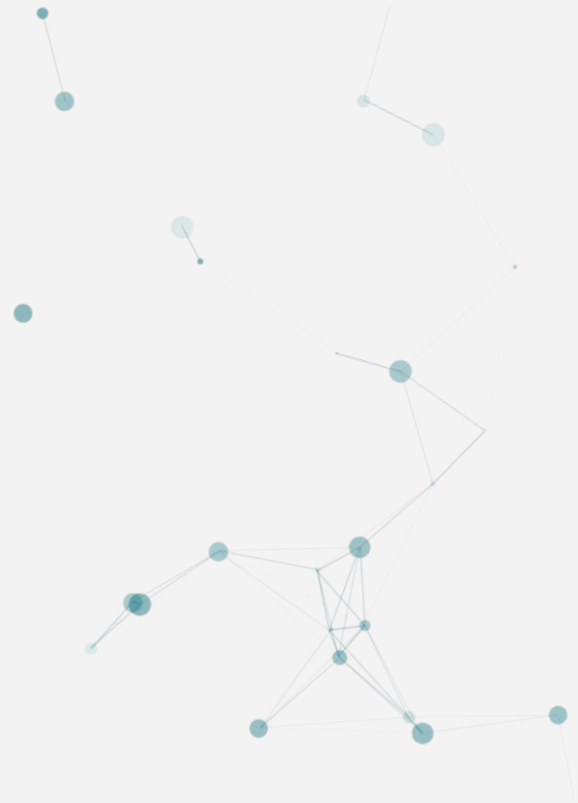


Network simulations with



Keefer Rourke

2020.11.24 - Guelph Coding Community

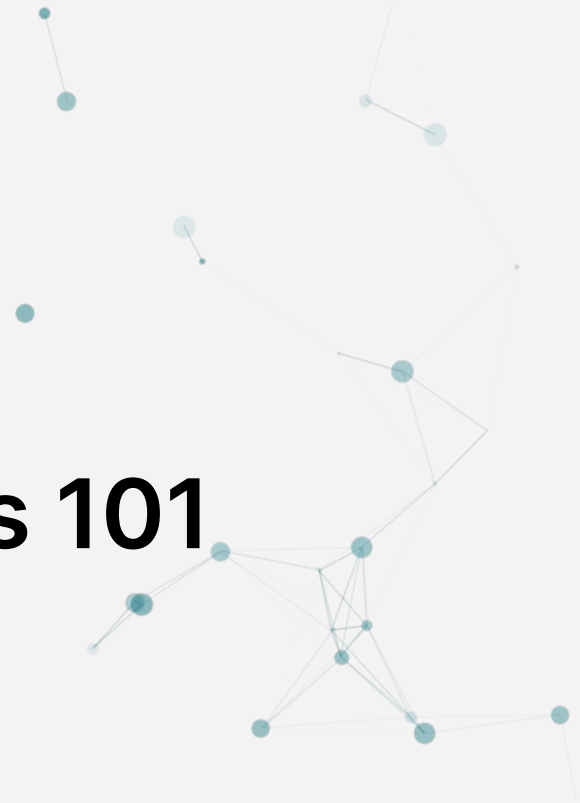


What we're going to cover

- The OSI layered network model
- Types of networks to consider
- WANETs and MANETS
- Specifying a simulation
- ns-3 programming with C++
- ns-3 models and classes
- Understanding simulation results
- Your first simulation
- Simulating MANETs
- Pro-tips
- Limitations and aggravations

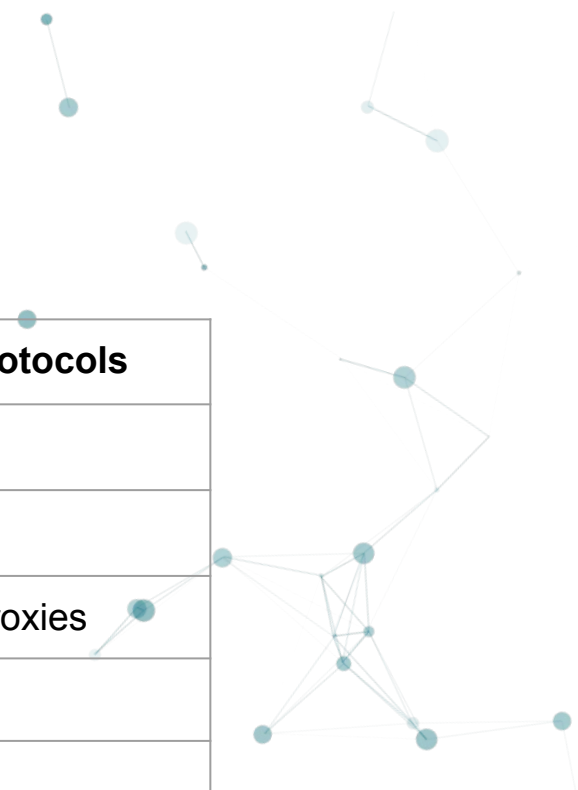


Computer Networks 101



The OSI network model

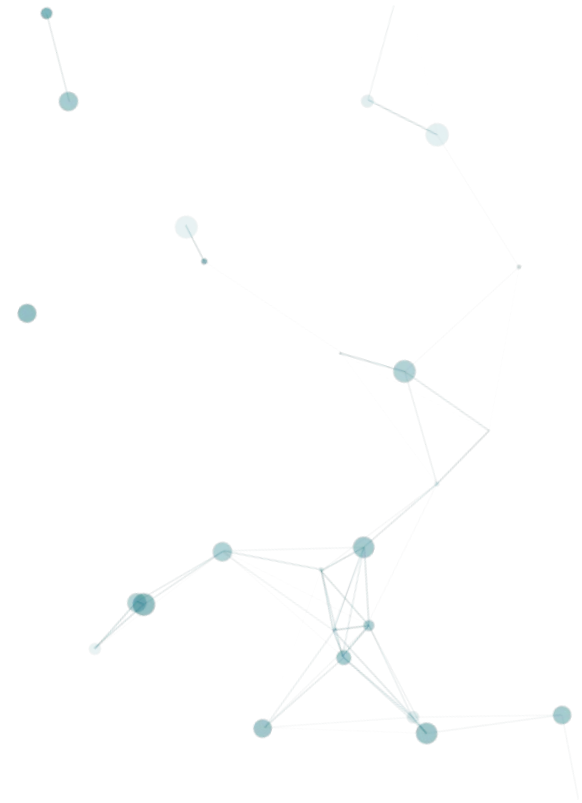
- Layers build on top of each other, are sometimes fuzzy

A decorative network diagram in the top right corner of the slide. It features several blue circular nodes connected by thin, light blue lines, forming a complex web-like structure. The nodes are scattered across the upper right portion of the image.

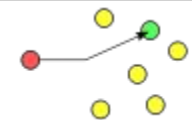



Layer	Purpose	Sample Protocols
Application	Useful user-facing stuff!	HTTP
Presentation	Translating data for use in applications	SSL/TLS
Session	Connection management	Sockets, proxies
Transport	Message segmentation/desegmentation	TCP, UDP
Network	Routing data between networks	IP, BGP
Data Link	Establishing topology	802.1 x MAC, ARP
Physical	Physical connections and signals	802.1x PHY

Types of networks

- Networks can be categorized in lots of ways
 - Fixed vs fluid infrastructure
 - Wired vs wireless
 - Physical vs virtual or overlay
 - Cellular networks
 - LAN vs WAN
 - NAT (network address translation) or other
 - Topology differences
 - Routing strategies
 - etc.



Types of routing

Type	What	Picture
unicast	One-to-one (selective) transmission between nodes.	
broadcast	One-to-all (non-selective) transmission between nodes.	
multicast	One-to-many (selective) transmission between nodes.	
anycast	One-to-one (of many) transmission between nodes.	



Wireless ad hoc networks

- Ad hoc networks refer to networks with:
 - Little to no infrastructure
 - Nodes that are directly connected to one another without intermediary routers
 - Distributed control
- May or may not require prior setup

Mobile ad hoc networks (MANETs)

- Refer to ad hoc networks with mobile nodes
- Dynamic/changing network topologies
- Self-governance and management of nodes and data

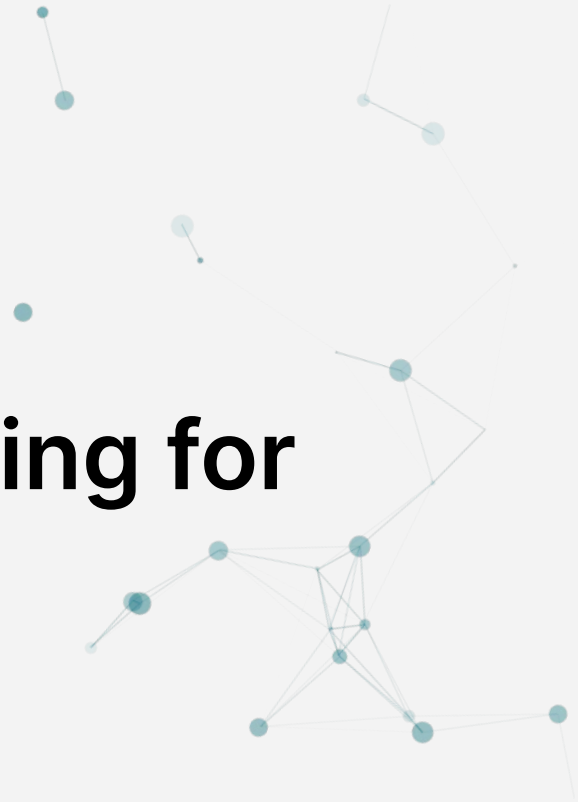


Physical properties of MANETs

- Homogeneous or heterogeneous hardware configurations
- Radio strength, battery capacity, bandwidth availability all real concerns
- Mobile nodes can complicate data transfer
 - Dropped connections
 - Changing routes
 - Data isolation/replication as networks form partitions
- Software solutions needed to make smart decisions given variance of hardware capabilities and node configurations



Simulation programming for beginners



Specifying a simulation

- Simulation area (dimensions and topography)
- Mobility of nodes (static, deterministic, random walk, random waypoint, etc.)
 - If something more complicated, can you compose the movement model from other models?
- Network hardware model
 - Connection speed
 - Radio channels
 - Signal propagation loss over area/time
- Formation of network topologies
 - How are MAC addresses assigned?
 - How are IP addresses assigned?
- RNG seed



ns-3 programming

Some things to keep in mind:

- ns-3 is a “next-generation” simulation under development since 2006
- The architecture is inspired by the Component Object Model (COM) pattern from .NET frameworks
 - (We’ll revisit this)
- You’re working with C++
 - Beware what’s on the stack and what’s on the heap
 - Beware of copy and move semantics



Aside: Modern C++ programming

- Prior to C++11, working with the language sucked
 - It didn't have smart pointers
 - It didn't have range-based for loops
 - It didn't have lambda expressions
- Before C++17, `std::` container types didn't even have uniform interfaces
- In 2020, the standard library is pretty good, and you might even enjoy working with C++ if the codebase is sufficiently new!
- But what about older codebases?



Aside: Modern C++ programming

- Sufficiently old projects developed their own offerings for modern C++ `std::` classes
- Firefox (1998), ns-3 (2006), and even QT (2011) each have their own custom implementations for strings, ref-counting pointers, and container classes
- Prepare to learn the “standard library” for these projects if you want to work with them



Getting started with ns-3

Step 1: Get it.

- ns-3 is distributed in a few ways:
 - via `bake.py`
 - via git repository
 - via tarball download



Getting started with ns-3

Step 1: Get it.

bake.py:

- Is for “advanced” users
- Can download, build, and install ns-3 and related software and addons
- Is really confusing
- We won't use it



Getting started with ns-3

Step 1: Get it.

By git:

- Provides a “basic” installation of ns-3
- Is all we need
- Is not a good way to get it* unless you are planning on hacking on ns-3 itself

*ns-3 encourages you to write your scripts directly in their source distributions... More on this later.



Getting started with ns-3

Step 1: Get it.

By tarball:

- Just the good stuff

```
$ wget https://www.nsnam.org/release/ns-allinone-3.32.tar.bz2
$ tar xjvf ns-allinone-3.32.tar.bz2
$ cd ns-allinone-3.32
$ python3 ./build.py --enable-examples
```



Getting started with ns-3

Step 2: Understand some weirdnesses

- If you're used to writing C/C++, you probably expect:
 - to link your driver code to compiled library dependencies
 - to keep your source code separated from that of your dependencies
- ns-3 devs want you to forget about how to structure C++ projects
 - Your projects now live in their project
 - All your code belongs under the scratch/ dir

typical c/c++ project

structure

```
my_project/  
├── build/  
├── src/  
├── include/  
├── dep-1/  
├── dep-2/  
└── CMakeLists.txt
```

ns-3 project structure

```
ns-allinone-3.32/  
└── ns-3.32/  
    ├── scratch/  
    │   └── your-src/  
    │       └── scratch-simulator.cc  
    ├── ...  
    ├── wscript  
    └── waf
```

It is misleading and unhelpful to call ns-3 simulations “scripts”.

... but we're going to do that anyway.



Getting started with ns-3

Step 2: Understand some weirdnesses

- waf is a build tool <<https://waf.io/book/>>
- ns-3 devs have extended waf beyond recognition to let you treat your C++ projects as “scripts”
- Always make sure that the top level waf binary is accessible to you (consider adding to your PATH)
- waf will only recognize *.cc and *.h files by default...


typical c/c++ project

structure

```
my_project/  
├── build/  
├── src/  
├── include/  
├── dep-1/  
├── dep-2/  
└── CMakeLists.txt
```

ns-3 project structure

```
ns-allinone-3.32/  
└── ns-3.32/  
    ├── scratch/  
    │   ├── your-src/  
    │   └── scratch-simulator.cc  
    ├── ...  
    ├── wscript  
    └── waf
```



Getting started with ns-3

Step 3: Build and run your first “script”

```
$ pwd
```

```
/home/user/project/ns-allinone-3.32/ns-3.32/
```

```
$ ./waf --run examples/tutorial/first
```

```
Waf: Entering directory
```

```
`/home/user/project/ns-allinone-3.32/ns-3.32/build'
```

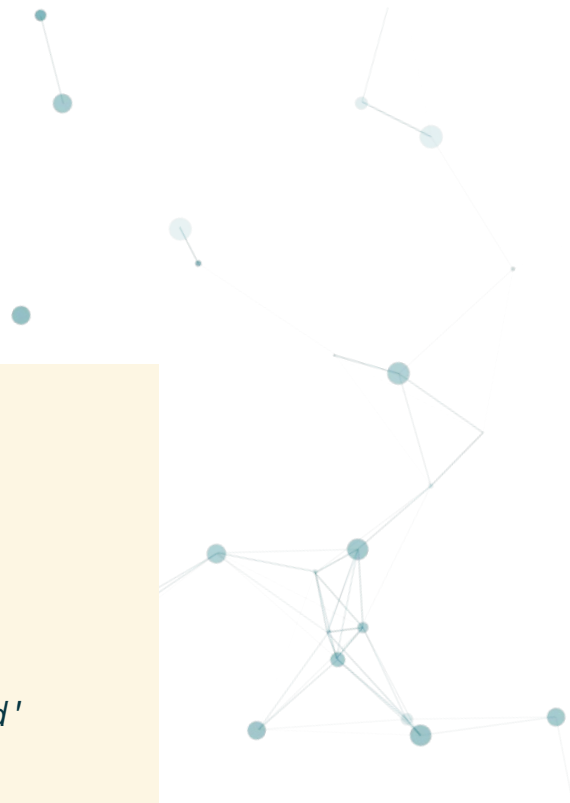
```
Waf: Leaving directory `/home/user/ns-allinone-3.32/ns-3.32/build'
```

```
Build commands will be stored in build/compile_commands.json
```

```
'build' finished successfully (1.369s)
```

```
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
```

```
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
```



Getting started with ns-3

Step 3: Build and run your first “script”

- ns-3 has some tutorial scripts
 - But they are really just a random collection of examples of some example network topologies
- Learn what scripts you can run:

```
$ ./waf --run does-not-exist 2>&1 | tr ',' '\n'
```

- Some script names are aliases for others...
- Mostly they correspond to source files in subdirs of ns-3.32/



So you want to write a simulation?

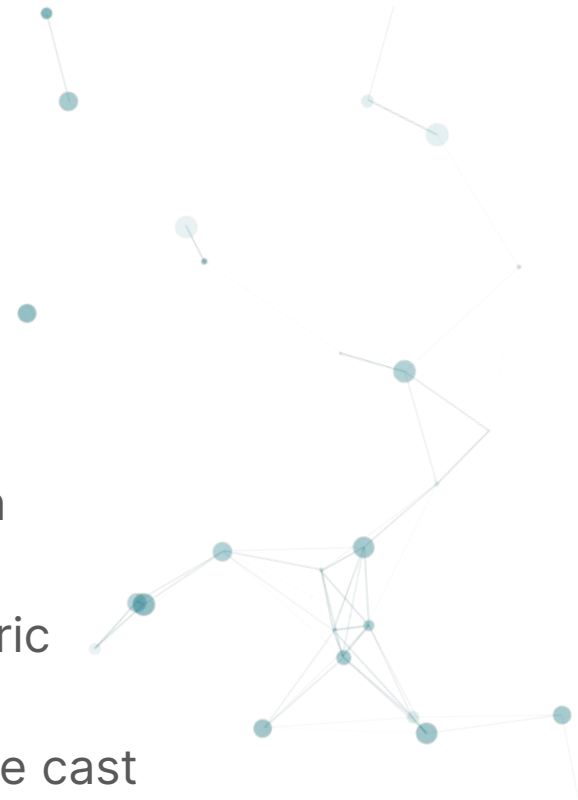


Aside: COM Programming in ns-3

The Component Object Model is:

- An **runtime** object oriented model/framework
- Suited to multiple language-bindings and IPC
- Types must be identifiable at runtime, so are each assigned a GUID
- All objects are ref-counted and implement a generic interface that supports this (typically, IUnknown)
- Classes implement one or more interfaces, and are cast by the `→QueryInterface<I>()` method
- Enforces separation of interface and implementation*

* except that ns-3 COM doesn't...



Aside: COM Programming in ns-3

Key ideas:

- Classes are meant to have a prototypical Typeld
 - Default class attributes are set through the Typeld
 - Attrs are ro/rw, have a string name, value, and help text
 - Factories create instances from the Typeld
- Refcounting pointers and factory methods:
 - `Ptr<Node> node = CreateObject<Node>()`
- It's supposed to be easy to reach into the guts of ns-3 to tweak values

<https://www.nsnam.org/docs/manual/html/attributes.html>



Aside: COM Programming in ns-3

Suppose you want to change the packet queue size of particular node in a network to see if there would be a performance impact...

```
// Set up code. Let's make a node with a Wifi device and packet queue.  
using namespace ns3;
```

```
Ptr<Node> n = CreateObject<Node>();  
Ptr<WifiNetDevice> net0 = CreateObject<WifiNetDevice>();  
n->AddDevice (net0);
```

```
Ptr<Queue<Packet>> q = CreateObject<DropTailQueue<Packet>>();  
net0->AddQueue(q);
```

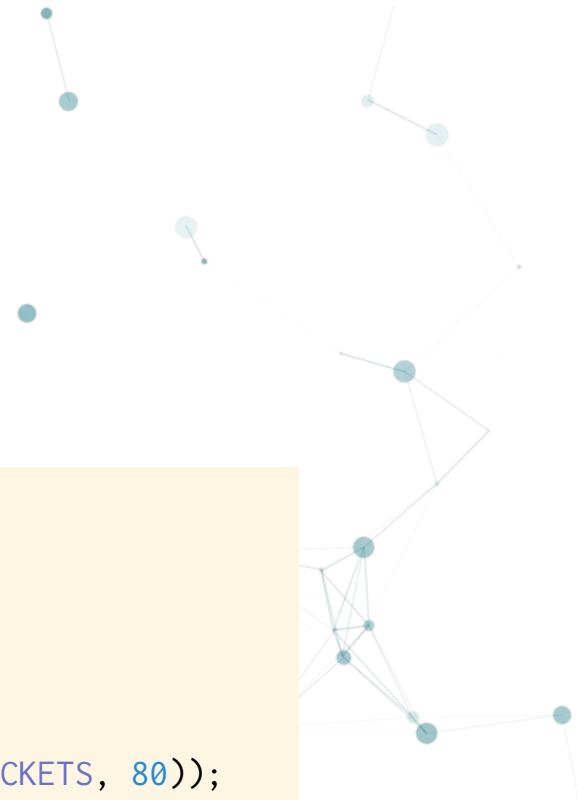


Aside: COM Programming in ns-3

Suppose you want to change the packet queue size of particular node in a network to see if there would be a performance impact...

```
// Retrieve the queue and set the attribute.
```

```
PointerValue ptr;  
net0->GetAttribute("TxQueue", ptr);  
Ptr<Queue<Packet>> txQueue = ptr.Get<Queue<Packet>>();  
txQueue->SetAttribute("MaxSize",  
    QueueSizeValue(QueueSize(QueueSizeUnit::PACKETS, 80)));
```

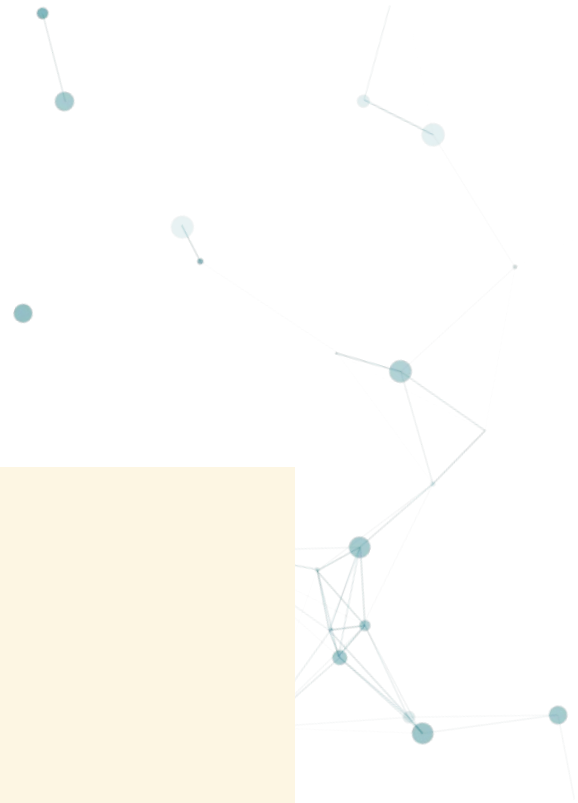


Aside: COM Programming in ns-3

Suppose you want to change the packet queue size of particular node in a network to see if there would be a performance impact...

```
// Retrieve the queue and set the attribute.
```

```
PointerValue ptr;  
net0->GetAttribute("TxQueue", ptr);  
Ptr<Queue<Packet>> txQueue = ptr.Get<Queue<Packet>>();  
txQueue->SetAttribute("MaxSize", StringValue("80p"));
```



Aside: COM Programming in ns-3

Suppose you want to change the packet queue size of particular node in a network to see if there would be a performance impact...

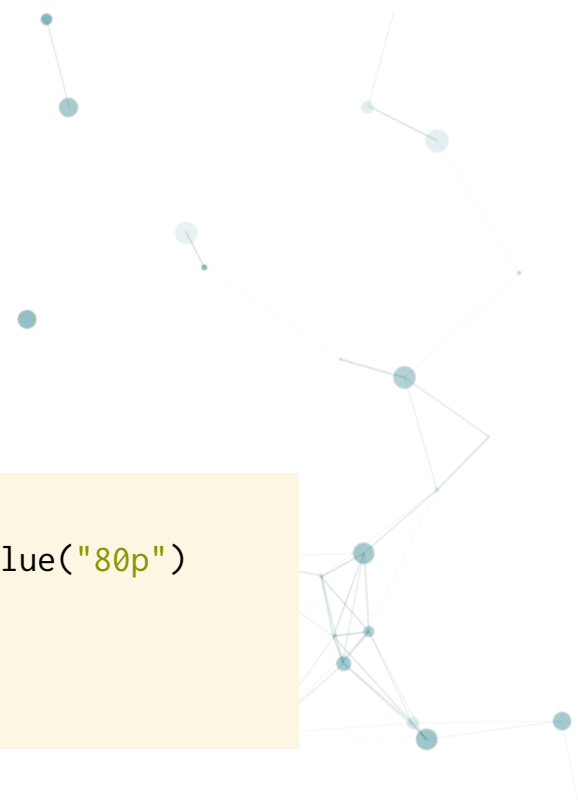
```
// Just set the attribute directly...?
```

```
Config::Set("/NodeList/0/DeviceList/0/TxQueue/MaxSize", StringValue("80p"))
```

```
// Even for all instances?
```

```
Config::Set("ns3::QueueBase::MaxSize", StringValue("80p"))
```

This is made possible by the runtime type-system, so you can save yourself some typing.

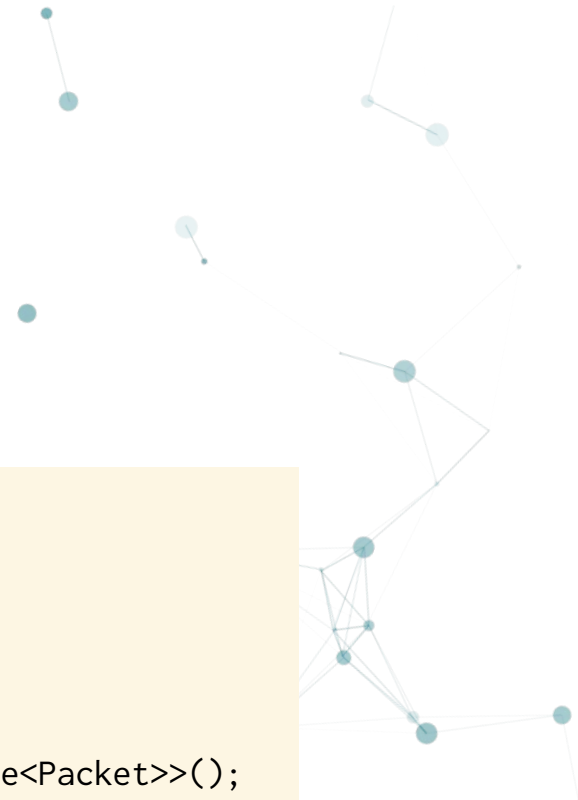


Aside: COM Programming in ns-3

Suppose you want to mess with something on a particular instance of a broader interface...

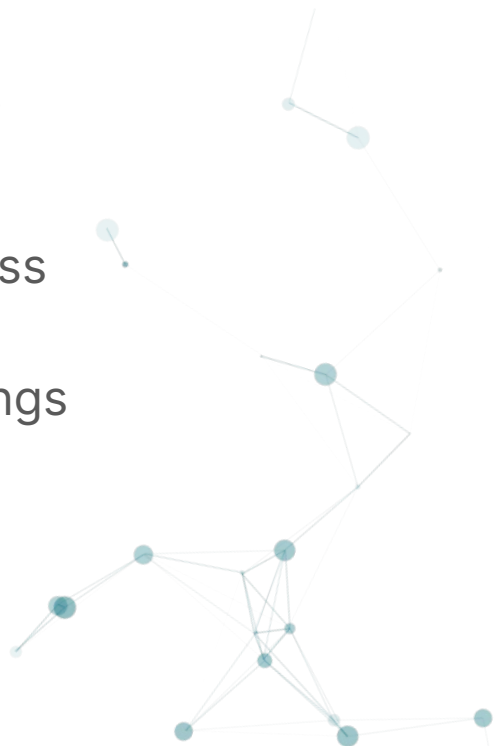
```
// Retrieve the queue and perform a safe downcast.
```

```
PointerValue ptr;  
net0->GetAttribute("TxQueue", ptr);  
Ptr<Queue<Packet>> txQueue = ptr.Get<Queue<Packet>>();  
  
Ptr<DropTailQueue<Packet>> dtq = txQueue->GetObject<DropTailQueue<Packet>>();  
NS_ASSERT(dtq != 0);
```



Aside: COM Programming in ns-3

- ns-3 architecture is mostly inspired by COM to address some deficiencies with ns-2's type system
- A lot of things are done with Typelds and special strings
- Key patterns to be aware of:
 - `NS_OBJECT_ENSURE_REGISTERED(Class);`
 - `auto x = ns3::CreateObject<T>();`
 - `x->SetAttribute("AttrName", ns3::AttributeValue(...));`
 - `Config::Set("container/indices/attr/attr/attr", value);`
- Some python bindings are incubating, but as of yet aren't quite useful



ns-3 models and classes

ns-3 has extensive class reference docs here:

- <https://www.nsnam.org/doxygen/index.html>

Broadly speaking, every part of the OSI Network Model is represented *somehow*.

Generally there are “helpers” which make it is bit easier to set up networks with a collection of nodes.



ns-3 models and classes

Nodes and NodeContainers

- `ns3::Node`
 - Network entity (e.g. router, modem, repeater, etc.)
 - Install and configure network devices on this
 - Define how these move around if you want
- `ns3::NodeContainer`
 - Holds a `std::vector<Ptr<Node>>`
 - Can be copied safely
 - Nodes can belong to multiple containers



ns-3 models and classes

Helper classes

- Usually designed to work with `NodeContainers`
- Configure a factory for instantiating objects
 - Network devices
 - Routing protocols
 - Network applications
 - Etc.
- Mostly work the same, but there's no Helper interface



Understanding simulation results



Animation and visualization

Two animation interfaces:

- NetAnim:
 - Standalone QT program
 - Can visualize simulations *after* they have run
 - Add support by constructing an AnimationInterface just before you start your simulation
- GraphViz:
 - Simulations have implicit bindings to PyGraphViz
 - Run with the --vis option to see visualizations *live* as your simulation runs
 - Buggy, doesn't show packet transmissions



Packet captures

- ns-3 can output pcap files
 - Compatible with Wireshark or tcpdump
- Some connection helper classes have methods like:
 - `EnablePcapAll(std::string)`
 - `EnablePcapIpv4All(std::string)`
- Disclaimer: I've never got this to work properly yet in any of my simulations



https://www.nsnam.org/docs/release/3.9/tutorial/tutorial_23.html

Your first simulation



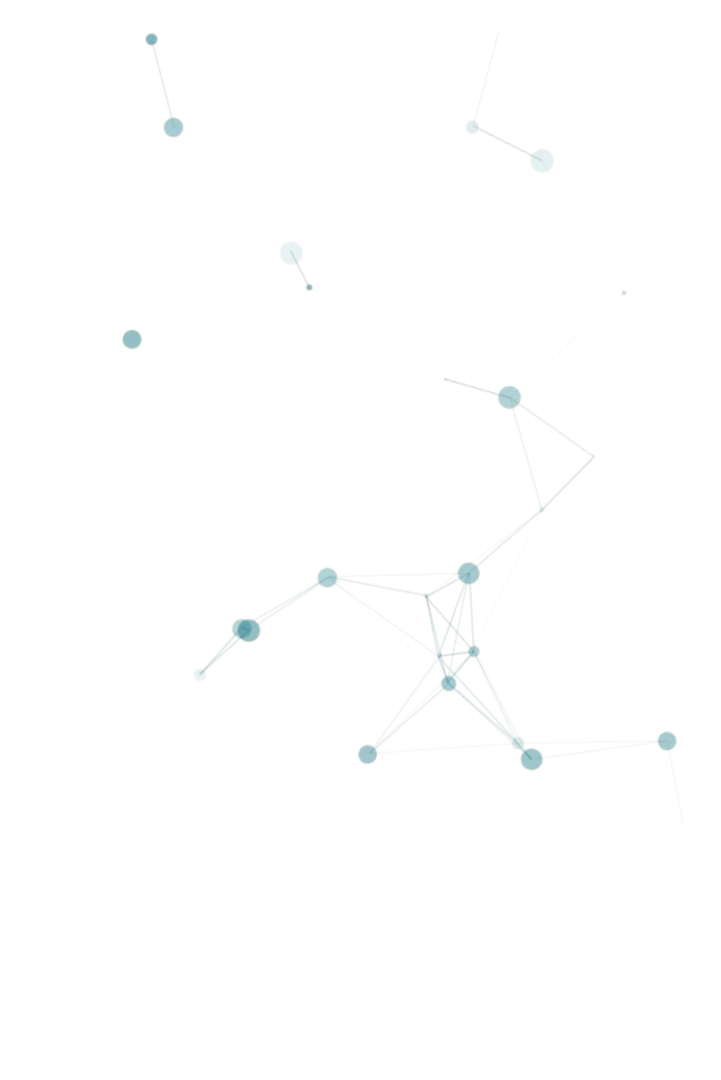
Recipe for a simulation

```
using namespace ns3;

int main(int argc, char* argv[]) {
    // Get options.
    CommandLine cmd;
    ...
    cmd.Parse(argc, argv);
    // Set up nodes.
    NodeContainer nodes;
    ...
    // Add the animation interface.
    AnimationInterface animation("anim-trace-file.xml");
    // Run the simulation.
    Simulator::Stop(Seconds(10));
    Simulator::Run();
    Simulator::Destroy();
    return EX_OK;
}
```



Demo time.



Simulating MANETs



Simulating MANETs

- Lots to consider
- Gillis Lab is studying:
 - Mobility models
 - Data replication and storage
 - Requirements for device density
 - Cluster formation and identification
 - Routing protocols
- Simulations help when you can't realistically get a bunch of devices operating on a mesh in a community



Mobility

Model is concerned with:

- Defining simulation area
- Setting node velocity (m/s)
- Initial allocation of nodes over area
- Defining patterns of movement over time
- Changing patterns of movement over time

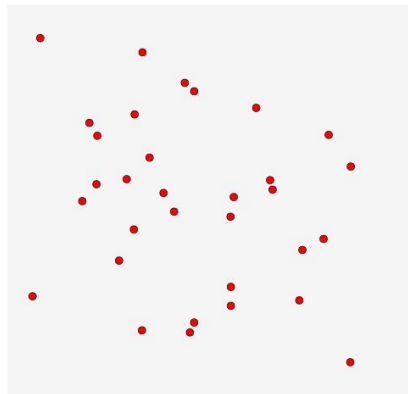
Some available models:

- ConstantPosition
- RandomWaypoint
- RandomWalk2d
- GaussMarkov (3D)
- Others...

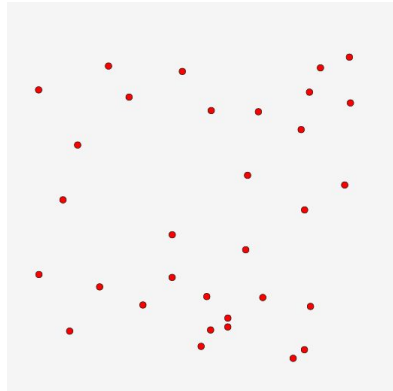
For the most part we are only interested in RandomWaypoint and RandomWalk2d.

Mobility models

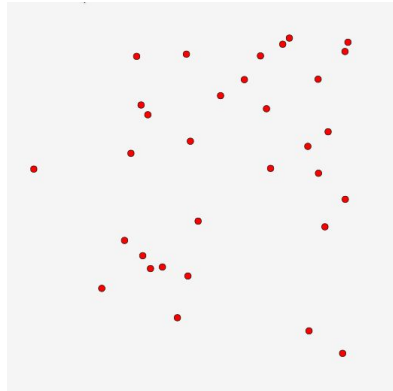
Some interesting examples.



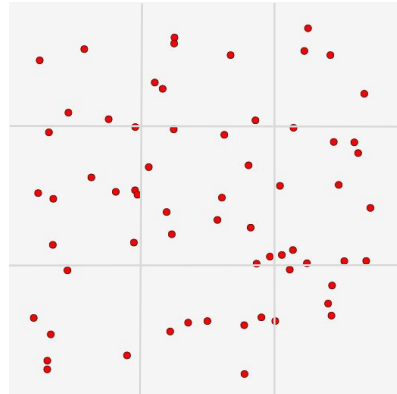
Random waypoint



Jitter
(RandomWalk2D default)



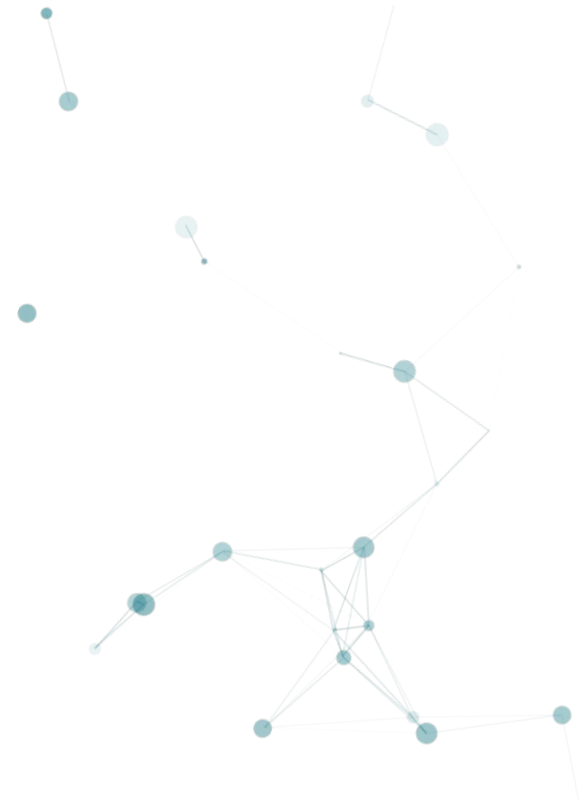
Walk over area
(RandomWalk2D)



Travellers between
partitions on 3×3 grid

Mobility models

Underlying implementation supports 3D movement, but the visualization tools **do not** appear to.



Routing

Various routing algorithms have been designed for use in MANETs and other distributed networks

- Proactive
 - Periodically deliver routing tables through the network
 - DSDV, OSLR
- Reactive
 - Finds routes based on user demand or other factors
 - Uses Route Request or Discovery packets
 - AODV
- Hybrid



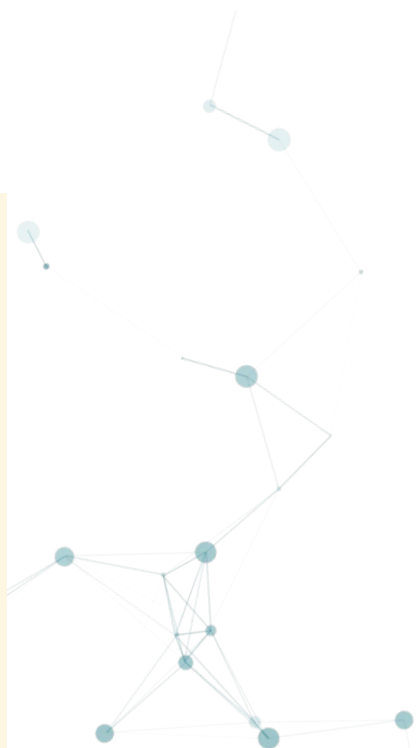
Recipe for ad hoc network simulations

```
using namespace ns3;

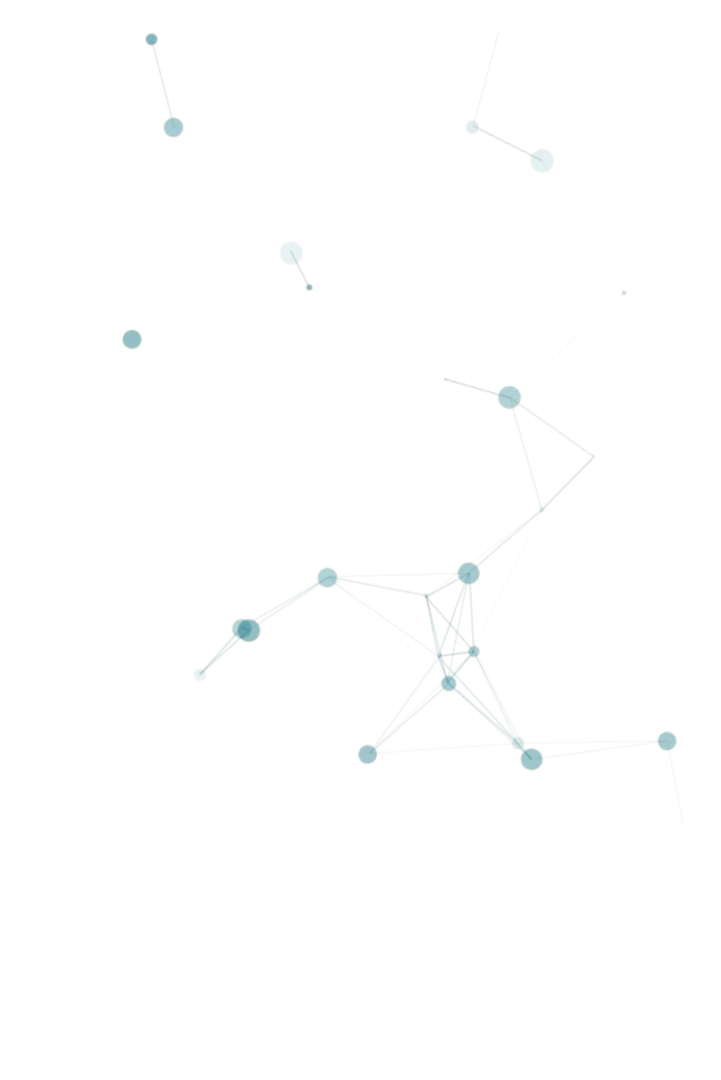
int main(int argc, char* argv[]) {
    ...
    YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default();
    wifiPhy.SetPcapDataLinkType(YansWifiPhyHelper::DLT_IEEE802_11_RADIO);
    auto wifiChannel = YansWifiChannelHelper::Default();
    wifiPhy.SetChannel(wifiChannel.Create());

    WifiMacHelper wifiMac;
    wifiMac.SetType("ns3::AdhocWifiMac");

    WifiHelper wifi;
    wifi.SetStandard(WIFI_STANDARD_80211b);
    ...
    return EX_OK;
}
```



Demo time.



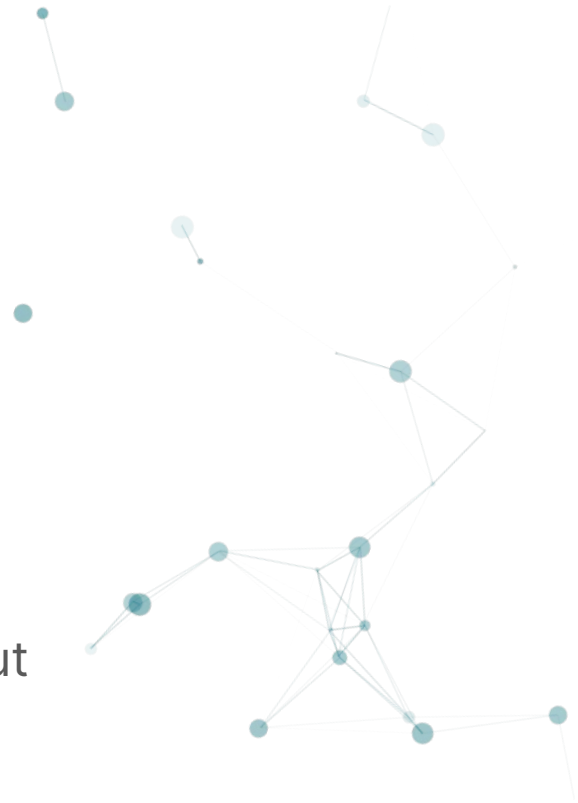
Pro tips

- Everything should be configurable
- Try to provide sane defaults
- Recompiling sucks, so make use of the CommandLine
- Always document your seed
- Use helpers and containers where possible
- Mobility is a good place to start, then add networking
- Start small and build up complexity
- Always use the latest ns-3



Limitations and aggravations

- ns-3 uses an entirely new/different style of programming to get used to
- Some constructors are very side-effect-y
- Generally terrible default values lead to weird behaviours
- There's a fair bit of global state to be aware of
- No convenient methods to query information about neighbor nodes in a topology
- Generally slow to develop and run
- Big simulations can really hurt your CPU
 - Consider having some capable cloud hardware available?



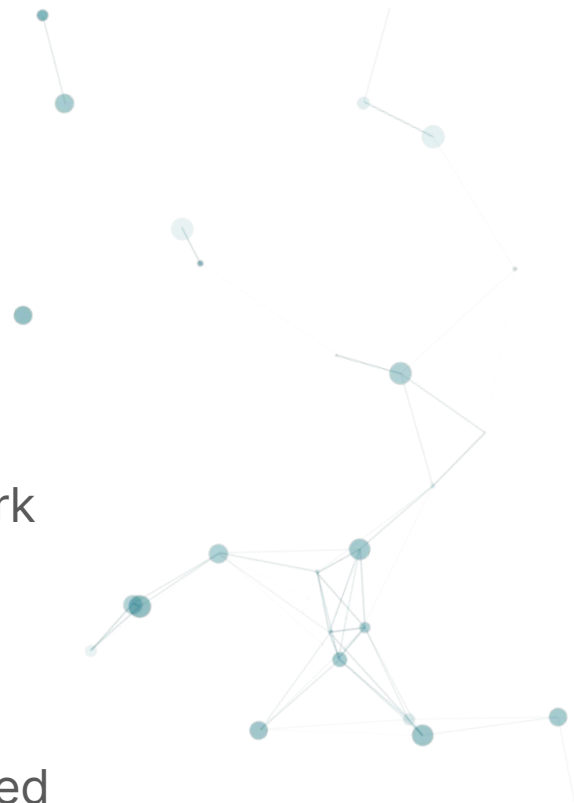
Buyer beware

Wide open-field

- There's a lot of research out there about MANETs
- Parts are studied together and in isolation
- Higher complexity → more interesting → more work

Simulation studies vary greatly

- A lot are difficult or impossible to reproduce
- Simulation parameters and RNG seeds not specified
- Source code not always available :(





Questions?



References

1. <https://osi-model.com/>
2. <https://www.nsnam.org/docs/architecture.pdf>
3. https://en.wikipedia.org/wiki/Component_Object_Model
4. <https://www.nsnam.org/docs/manual>

