
ns-3 Training

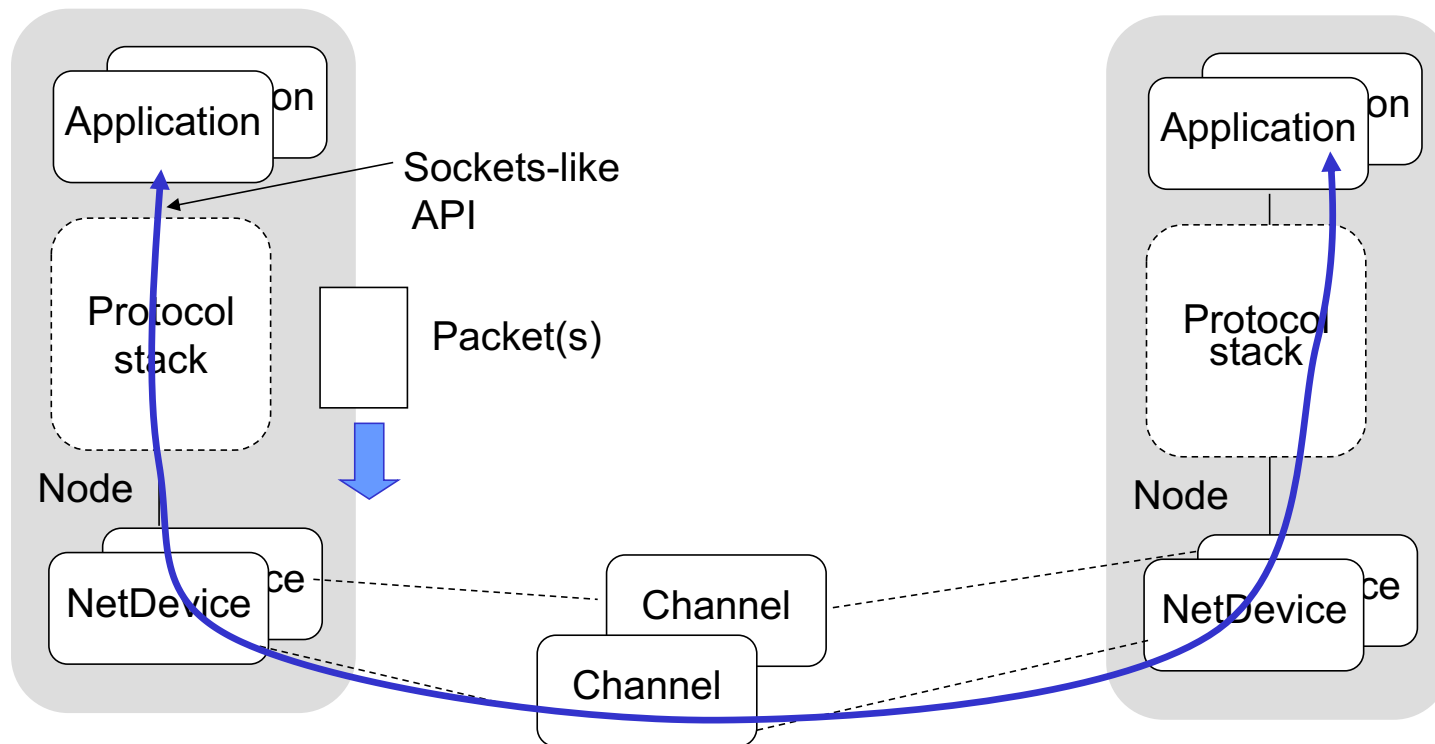
Program Structure and Simulation Campaigns

Example walkthrough

- This section progressively builds up a simple ns-3 example, explaining concepts along the way
- Files for these programs are available on the ns-3 wiki

Example program

- `wns3-version1.cc`
 - Link found on wiki page
 - Place program in scratch/ folder



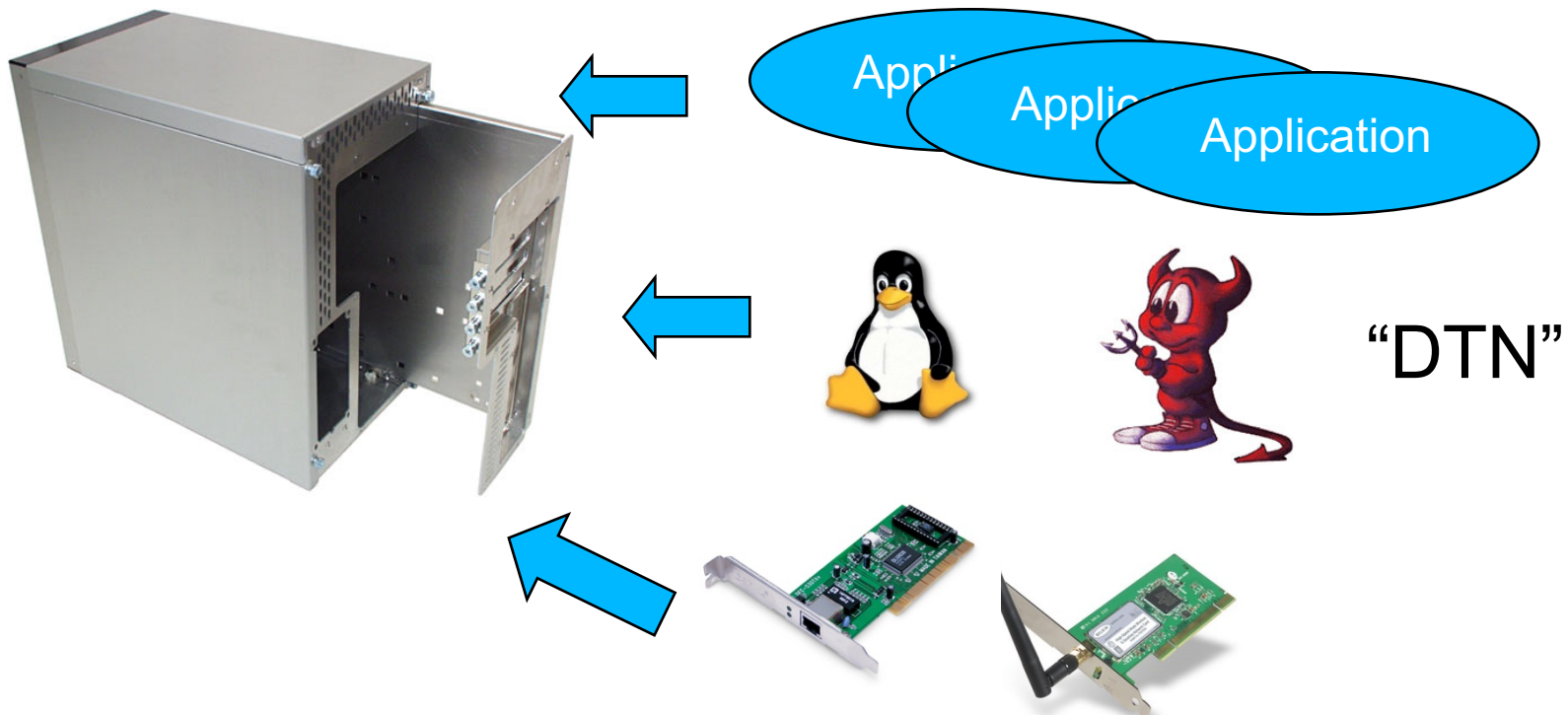
Fundamentals

Key objects in the simulator are Nodes,
Packets, and Channels

Nodes contain Applications, “stacks”, and
NetDevices

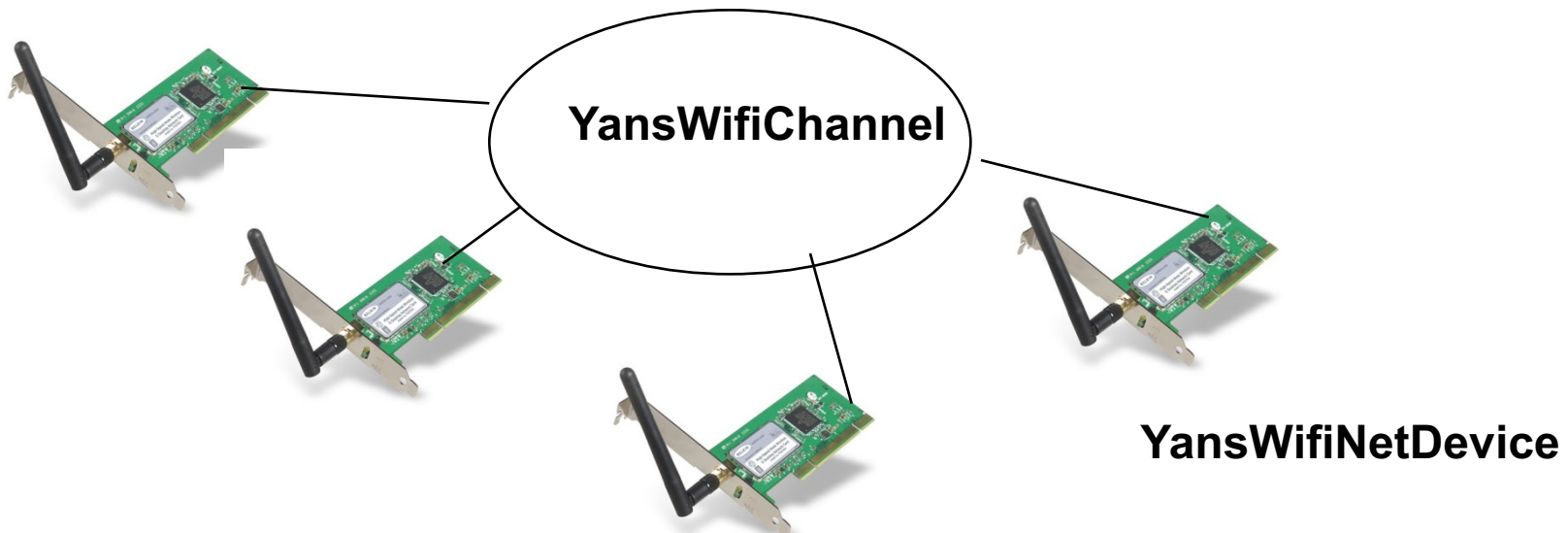
Node basics

A Node is a shell of a computer to which applications, stacks, and NICs are added



NetDevices and Channels

(Originally) NetDevices were strongly bound to Channels of a matching type



- ns-3 Spectrum models relax this assumption

Nodes are architected for multiple interfaces

Internet Stack

- Internet Stack
 - Provides IPv4 and some IPv6 models currently
- No non-IP stacks ns-3 existed until 802.15.4 was introduced in ns-3.20
 - but no dependency on IP in the devices, Node object, Packet object, etc. (partly due to the object aggregation system)

Other basic models in ns-3

- Devices
 - WiFi, WiMAX, CSMA, Point-to-point, ...
- Error models and queues
- Applications
 - echo servers, traffic generator
- Mobility models
- Packet routing
 - OLSR, AODV, DSR, DSDV, Static, Nix-Vector, Global (link state)

Structure of an ns-3 program

```
int main (int argc, char *argv[])
{
    // Set default attribute values

    // Parse command-line arguments

    // Configure the topology; nodes, channels, devices, mobility

    // Add (Internet) stack to nodes

    // Configure IP addressing and routing

    // Add and configure applications

    // Configure tracing

    // Run simulation

    // Handle any post-simulation data processing
}
```

Helper API

- The ns-3 “helper API” provides a set of classes and methods that make common operations easier than using the low-level API
- Consists of:
 - container objects
 - helper classes
- The helper API is implemented using the low-level API
- Users are encouraged to contribute or propose improvements to the ns-3 helper API

Containers

- Containers are part of the ns-3 “helper API”
- Containers group similar objects, for convenience
 - They are often implemented using C++ std containers
- Container objects also are intended to provide more basic (typical) API

The Helper API (vs. low-level API)

- Is not generic
- Does not try to allow code reuse
- Provides simple 'syntactical sugar' to make simulation scripts look nicer and easier to read for network researchers
- Each function applies a single operation on a "set of same objects"
- A typical operation is "Install()"

Helper Objects

- NodeContainer: vector of Ptr<Node>
- NetDeviceContainer: vector of Ptr<NetDevice>
- InternetStackHelper
- WifiHelper
- MobilityHelper
- OlsrHelper
- ... Each model provides a helper class

Installation onto containers

- Installing models into containers, and handling containers, is a key API theme

```
NodeContainer c;  
c.Create (numNodes);  
...  
mobility.Install (c);  
...  
internet.Install (c);  
...
```

Native IP models

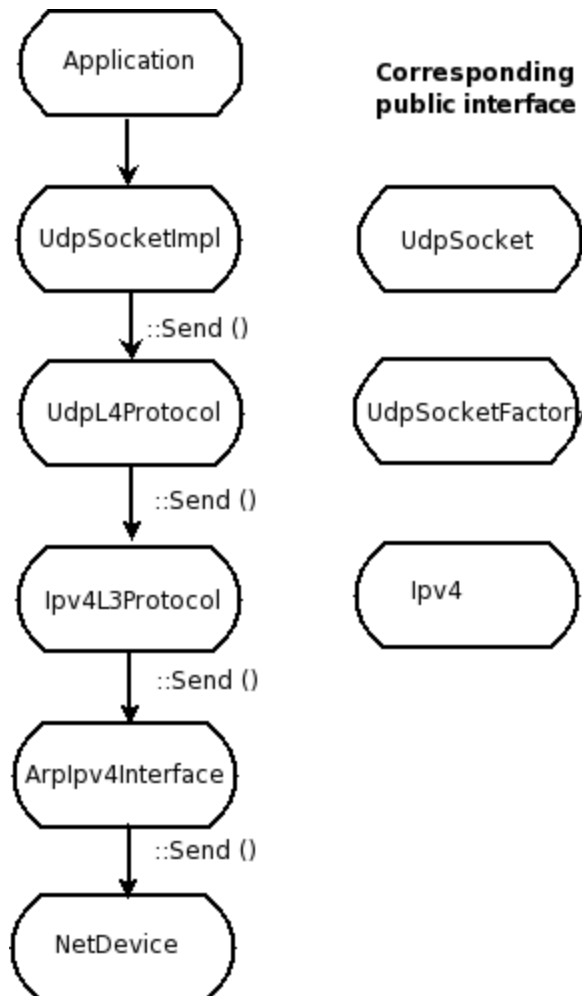
- IPv4 stack with ARP, ICMP, UDP, and TCP
- IPv6 with ND, ICMPv6, IPv6 extension headers, TCP, UDP
- IPv4 routing: RIPv2, static, global, NixVector, OLSR, AODV, DSR, DSDV
- IPv6 routing: RIPng, static

IP address configuration

- An Ipv4 (or Ipv6) address helper can assign addresses to devices in a NetDevice container

```
Ipv4AddressHelper ipv4;  
ipv4.SetBase ("10.1.1.0", "255.255.255.0");  
csmaInterfaces = ipv4.Assign (csmaDevices);  
  
...  
  
ipv4.NewNetwork (); // bumps network to 10.1.2.0  
otherCsmaInterfaces = ipv4.Assign (otherCsmaDevices);
```


Internet stack



- The public interface of the Internet stack is defined (abstract base classes) in `src/network/model` directory
- The intent is to support multiple implementations
- The default ns-3 Internet stack is implemented in `src/internet-stack`

Review of sample program (cont.)

```
ApplicationContainer apps;  
OnOffHelper onoff ("ns3::UdpSocketFactory",  
                  InetSocketAddress ("10.1.2.2", 1025));  
onoff.SetAttribute ("OnTime", StringValue ("Constant:1.0"));  
onoff.SetAttribute ("OffTime", StringValue ("Constant:0.0"));  
apps = onoff.Install (csmaNodes.Get (0));  
apps.Start (Seconds (1.0));  
apps.Stop (Seconds (4.0));
```

Traffic generator

```
PacketSinkHelper sink ("ns3::UdpSocketFactory",  
                      InetSocketAddress ("10.1.2.2", 1025));  
apps = sink.Install (wifiNodes.Get (1));  
apps.Start (Seconds (0.0));  
apps.Stop (Seconds (4.0));
```

Traffic receiver

Applications and sockets

- In general, applications in ns-3 derive from the `ns3::Application` base class
 - A list of applications is stored in the `ns3::Node`
 - Applications are like processes
- Applications make use of a sockets-like API
 - `Application::Start()` may call `ns3::Socket::SendMsg()` at a lower layer

Sockets API

Plain C sockets

```
int sk;
sk = socket(PF_INET, SOCK_DGRAM, 0);

struct sockaddr_in src;
inet_pton(AF_INET, "0.0.0.0", &src.sin_addr);
src.sin_port = htons(80);
bind(sk, (struct sockaddr *) &src,
      sizeof(src));

struct sockaddr_in dest;
inet_pton(AF_INET, "10.0.0.1", &dest.sin_addr);
dest.sin_port = htons(80);
sendto(sk, "hello", 6, 0, (struct
sockaddr *) &dest, sizeof(dest));

char buf[6];
recv(sk, buf, 6, 0);
}
```

ns-3 sockets

```
Ptr<Socket> sk =
udpFactory->CreateSocket ();

sk->Bind (InetSocketAddress (80));

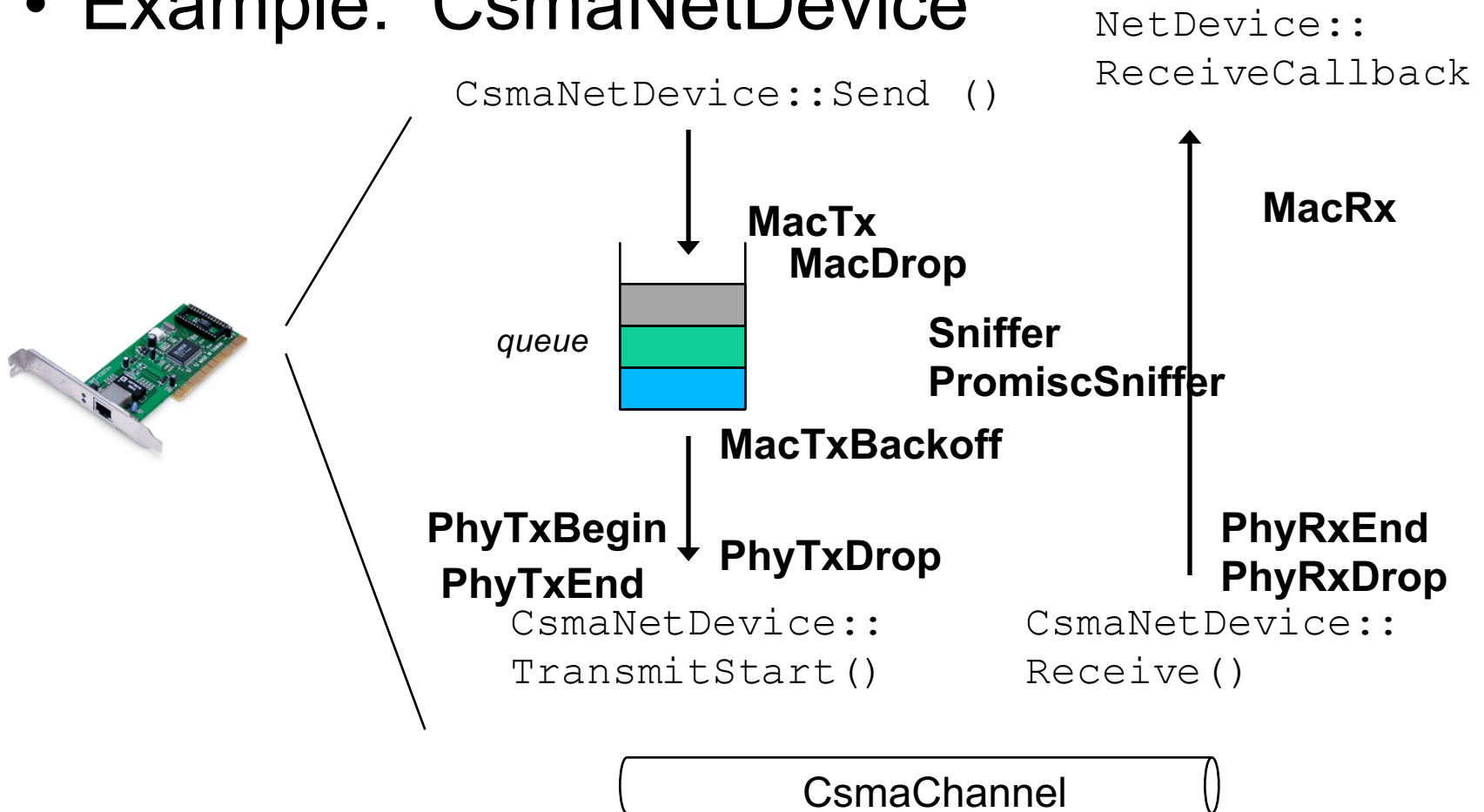
sk->SendTo (InetSocketAddress (Ipv4Address
("10.0.0.1"), 80), Create<Packet>
("hello", 6));

sk->SetReceiveCallback (MakeCallback
(MySocketReceive));
• [...] (Simulator::Run ())

void MySocketReceive (Ptr<Socket> sk,
Ptr<Packet> packet)
{
...
}
```

NetDevice trace hooks

- Example: CsmaNNetDevice



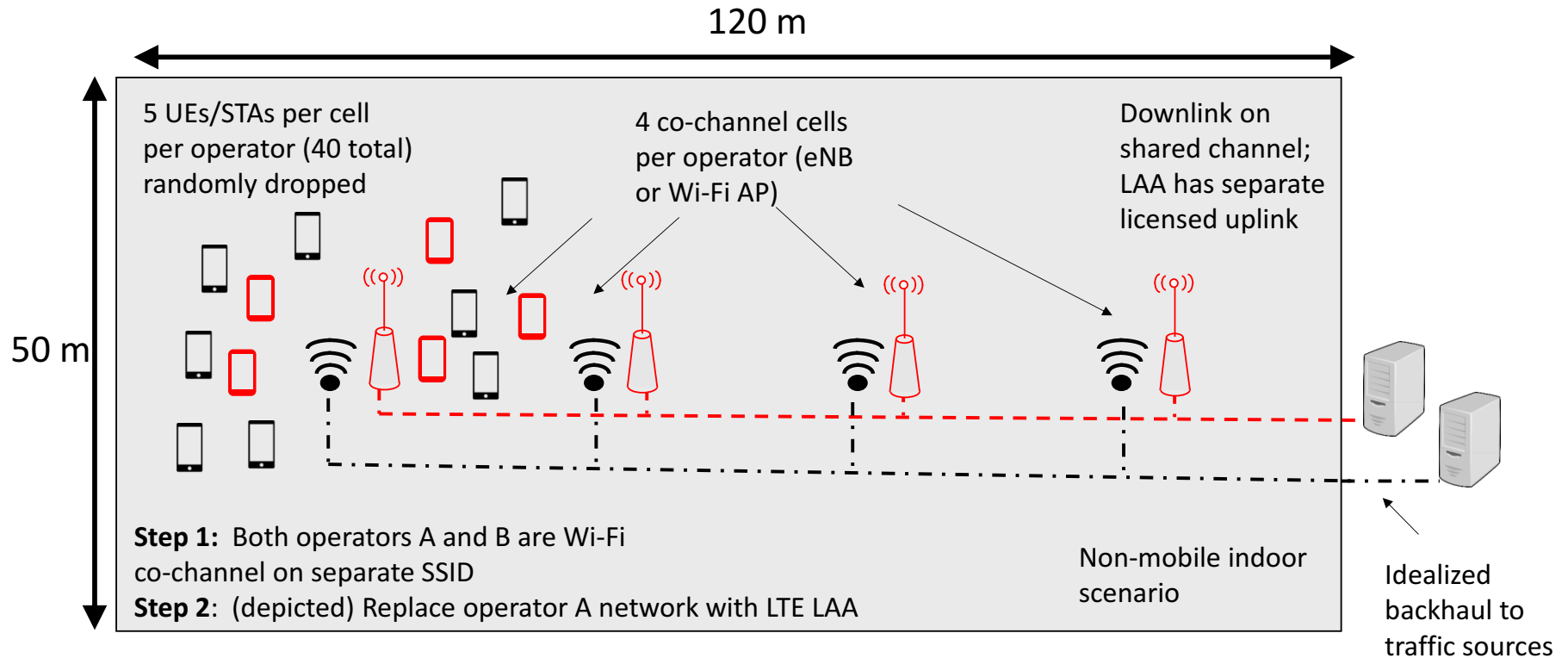
LTE/Wi-Fi Coexistence

case study

Use case: LAA Wi-Fi Coexistence

- ns-3 has been extended to support scenarios for LTE LAA/Wi-Fi Coexistence
- Methodology defined in 3GPP Technical Report TR36.889
- Enhancements needed:
 - Wireless models (LBT access manager, SpectrumWifiPhy, propagation/fading models)
 - Scenario support (traffic models)
 - Output data processing

Indoor 3GPP scenario



Indoor scenario details

Unlicensed channel model	3GPP TR 36.889	ns-3 implementation
Network Layout	Indoor scenario	Indoor scenario
System bandwidth	20 MHz	20 MHz
Carrier frequency	5 GHz	5 GHz (channel 36, tunable)
Number of carriers	1, 4 (to be shared between two operators) 1 for evaluations with DL+UL Wi-Fi coexisting with DL-only LAA	1 for evaluations with DL+UL Wi-Fi coexisting with DL-only LAA
Total Base Station (BS) transmission power	18/24 dBm	18/24 dBm Simulations herein consider 18 dBm
Total User equipment (UE) transmission power	18 dBm for unlicensed spectrum	18 dBm
Distance dependent path loss, shadowing and fading	ITU InH	802.11ax indoor model
Antenna pattern	2D Omni-directional	2D Omni-directional
Antenna height	6 m	6 m (LAA, not modelled for Wi-Fi)
UE antenna height	1.5 m	1.5 m (LAA, not modelled for Wi-Fi)
Antenna gain	5 dBi	5 dBi
UE antenna gain	0 dBi	0 dBi
Number of UEs	10 UEs per unlicensed band carrier per operator for DL-only 10 UEs per unlicensed band carrier per operator for DL-only for four unlicensed carriers. 20 UEs per unlicensed band carrier per operator for DL+UL for single unlicensed carrier. 20 UEs per unlicensed band carrier per operator for DL+UL Wi-Fi coexisting with DL-only LAA	Supports all the configurations in TR 36.889. Simulations herein consider the case of 20 UEs per unlicensed band carrier per operator for DL LAA coexistence evaluations for single unlicensed carrier.
UE Dropping	All UEs should be randomly dropped and be within coverage of the small cell in the unlicensed band.	Randomly dropped and within small cell coverage.
Traffic Model	FTP Model 1 and 3 based on TR 36.814 FTP model file size: 0.5 Mbytes. Optional: VoIP model based on TR36.889	FTP Model 1 as in TR36.814. FTP model file size: 0.5 Mbytes Voice model: DL only
UE noise figure	9 dB	9 dB
Cell selection	For LAA UEs, cell selection is based on RSRP (Reference Signal Received Power). For Wi-Fi stations (STAs), cell selection is based on RSS (Received signal power strength) of WiFi Access Points (APs). RSS threshold is -82 dBm. For the same operator, the network can be synchronized. Small cells of different operators are not synchronized.	RSRP for LAA UEs and RSS for Wi-Fi STAs
Network synchronization		Small cells are synchronized, different operators are not synchronized.

Outdoor 3GPP scenario

Outdoor layout: hexagonal macrocell layout. 7 macro sites and 3 cells per site. 1 Cluster per cell. 4 small cells per operator per cluster, uniformly dropped. ITU UMi channel model.

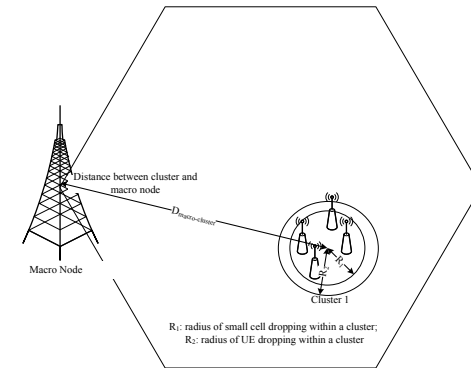
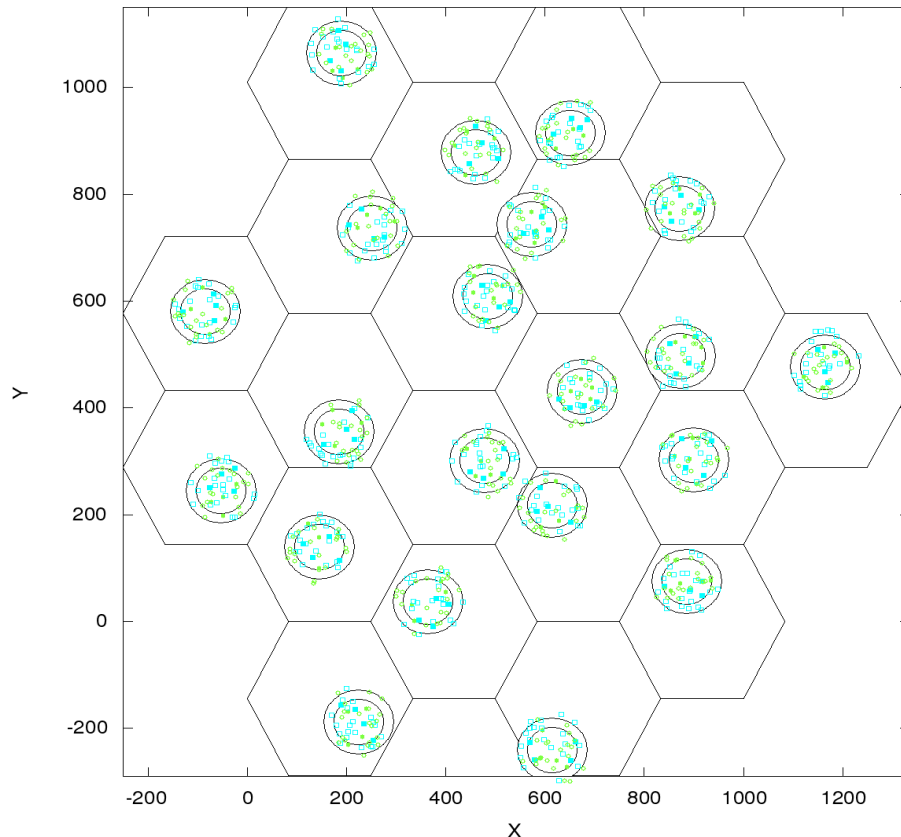
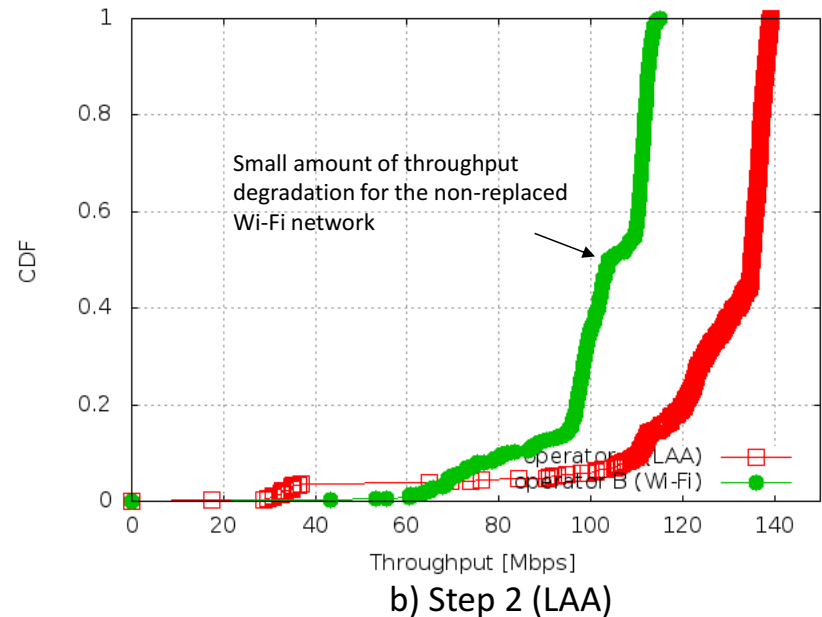
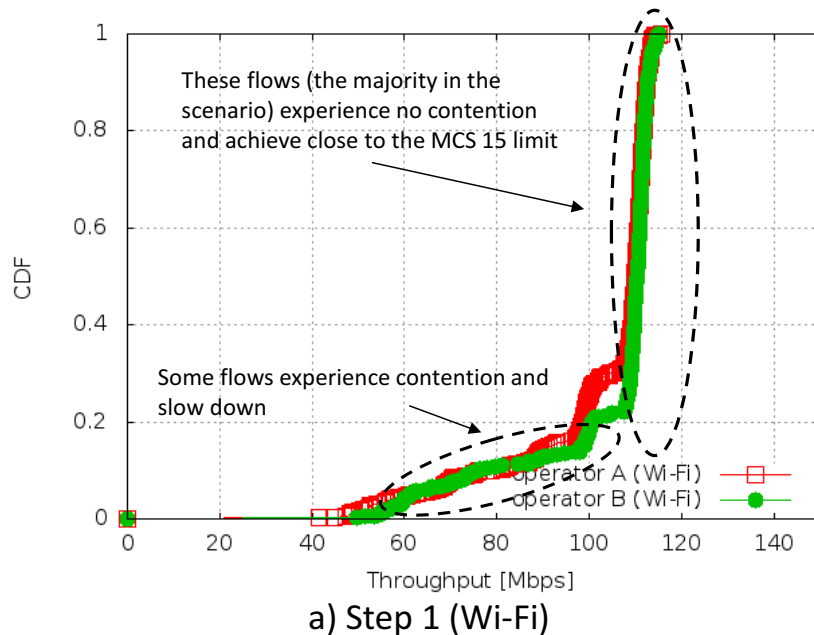


Figure source: 3GPP TR 36.889 V13.0.0(2015-05)

References

- ns-3 Wiki page:
 - <https://www.nsnam.org/wiki/LAA-WiFi-Coexistence>
 - module documentation
 - references to various publications
 - documentation on reproducing results
- Code:
 - <http://code.nsnam.org/laa/ns-3-lbt>

Sample results



Gnuplot

- `src/tools/gnuplot.{cc,h}`
- C++ wrapper around gnuplot
- classes:
 - Gnuplot
 - GnuplotDataset
 - Gnuplot2dDataset, Gnuplot2dFunction
 - Gnuplot3dDataset, Gnuplot3dFunction

Enabling gnuplot for your code

- examples/wireless/wifi-clear-channel-cmu.cc

```
CommandLine cmd;  
cmd.Parse (argc, argv);  
  
Gnuplot gnuplot = Gnuplot ("clear-channel.eps");  
for (uint32_t i = 0; i < modes.size (); i++)  
{  
    std::cout << modes[i] << std::endl;  
    Gnuplot2dDataset dataset (modes[i]);
```

produce a plot file that
will generate an EPS figure

one dataset per mode

```
    uint32_t pktsRecvd = experiment.Run (wifi, wifiPhy, wifiMac, wifiChannel);  
    dataset.Add (rss, pktsRecvd);  
}  
  
gnuplot.AddDataset (dataset);
```

Add data to dataset

Add dataset to plot

Matplotlib

- `src/core/examples/sample-rng-plot.py`

```
# Demonstrate use of ns-3 as a random number generator integrated  
# plotting tools; adapted from Gustavo Carneiro's ns-3 tutorial
```

```
import numpy as np  
import matplotlib.pyplot as plt  
import ns.core
```

```
# mu, var = 100, 225  
rng = ns.core.NormalVariable(100.0, 225.0)  
x = [rng.GetValue() for t in range(10000)]
```

```
# the histogram of the data  
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)
```

```
plt.title('ns-3 histogram')  
plt.text(60, .025, r'$\mu=100, \sigma=15$')  
plt.axis([40, 160, 0, 0.03])  
plt.grid(True)  
plt.show()
```

