Validation of Wi-Fi network simulation on ns-3

University of Washington Technical Report: August 2017

Rohan Patidar
Sumit Roy, Thomas R. Henderson, Morteza Mehrnoush

University of Washington
Seattle, WA 98195

August 7, 2017

1. Introduction

Network simulation is a powerful tool for performance evaluation of computer communications networks. A large number of tools are available commercially and as open-source, with different levels of abstraction, and with different levels of support for modeling various technologies. The ns-3 discrete-event network simulator is a popular open-source simulator used for networking research, and contains detailed models of the medium access control (MAC) layers of Wi-Fi and LTE. This paper is concerned with the validation of a crucial component of the Wi-Fi MAC protocol model in ns-3.

A key component of the Wi-Fi MAC model is the implementation of the contention resolution process known as the distributed coordination function (DCF). The DCF is a distributed process designed with the goal of achieving fair medium sharing among similar devices, despite the lack of any centralized scheduling control of the medium. In a distributed channel access process based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), and in a network with multiple stations, stations with data to send will inevitably attempt to transmit at the same time, resulting in an overlapping, failed transmission known as a collision. The DCF is designed to reduce the collision probability in such situations, in a manner that should, over time, result in successful transmissions and a relatively equal (fair) sharing of the medium between contending stations.

In its most basic form, the DCF consists of a carrier sensing (CS) function and a random backoff protocol. Carrier sensing allows stations to realize when the medium is busy, and to postpone transmission attempts until the medium is free. The carrier sense may be physical (detection of energy on the channel) and may also be virtual (detection of a reservation of the medium for a given period of time through protocol mechanisms). Once the medium becomes free, if two or more stations have a frame to send, the possibility exists that their transmissions will immediately collide. To avoid collisions, stations will backoff their attempts for a random number of slot times, based on a random variate selected from a contention window. If competing stations select different random variates, one station will access the medium first and will begin transmission, and the other stations will sense the medium has become busy and wait for the next idle opportunity. If two or more competing stations select the same transmission time, their transmissions will collide, and they will detect this based on the lack of a positive acknowledgement of the frame. Stations that collide will temporarily increase their contention window size and try again, and reset their contention window upon eventual successful transmission.

Consider the case of a dense network, with perhaps dozens of stations within detection range of one another, in which all stations have a saturating load (i.e., each station is always trying to transmit a new data frame). A key question is how the network behaves in such a saturating condition; will it result in a fair allocation of the medium, an unfair allocation with some stations dominating the channel access, or will the DCF mechanism become hopelessly deadlocked due to too many competing stations always vying for access? In a well-cited work published by Bianchi [1] and later updated by Tinnirello, Bianchi, and Xiao [4], the authors proved, with a Markov-chain analytical model and with network simulations, that the DCF mechanism will converge to a stable and fair allocation of the medium under a saturating load, and provided throughput performance predictions as a function of the number of competing stations.

The ns-3 simulator contains implementations of the DCF and a prioritized access control mechanism known as the enhanced distributed channel access (EDCA). However, a careful study of whether the ns-3 implementation can reproduce the results of Bianchi has never been been published. In 2012, Pei provided an initial attempt to validate the ns-3 802.11g and 802.11a OFDM model against the Bianchi results [3], but the work provided only preliminary data and some discrepancies between the model and the 802.11a ns-3 simulations remained unexplained. In this work, we provide a more thorough validation of the 802.11a OFDM ns-3 simulation model under a saturating load, and compare with an analytical model of the DCF based on [2]. We provide results for all 802.11a data rates, we explain the mathematical model employed, and make our simulation programs publicly available for others to reproduce.

In the next section, we briefly describe the DCF mathematical model proposed initially in [1] and later modified in [2], as per IEEE 802.11 specifications.

2. Mathematical Model

The DCF can be modeled as 2-D Markov chain according to Bianchi's work [1]. With the update in IEEE specifications, the Markov model presents [4] more relevant model which explains effect of finite packets retry. Further [2] presents more germane modeling for 11a in figure 2.1.

In saturation conditions, every station always has a frame available for transmission after the completion of each successful transmission. Each frame needs to wait for a random backoff time before transmitting. The backoff is performed in discrete time units called slots and the stations are synchronized on the slot boundaries; hence the time can be discretized in the analytical model.

The maximum value of the backoff counter of each station depends also on its transmission history (e.g., how many retransmissions the head-of-line frame has suffered). The stochastic process of the backoff counter evolution follows non-Markovian behavior, but when we assume that at each transmission attempt, the frame collision probability is constant and independent probability p (conditional collision probability), the model with state represented by the backoff stage and backoff counter can be modeled as Markovian as shown in figure 2.1.

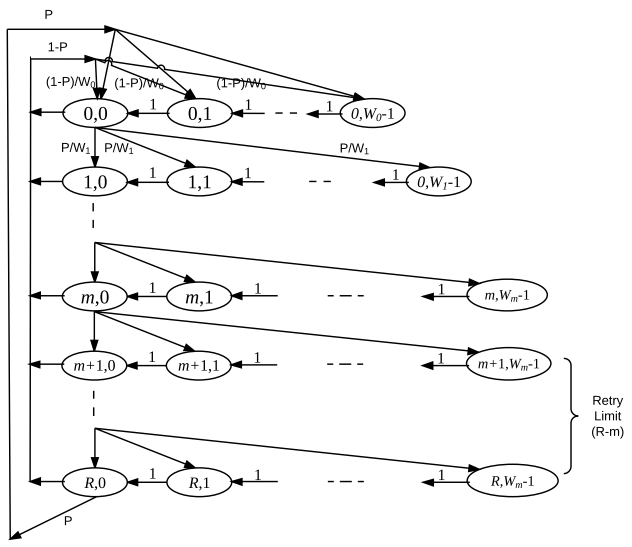

Figure 2.1.: Markov model for DCF

Using this Markov model, the probability that a station transmits in a randomly chosen slot time can be obtained as given by Equation 2.1:

$$\tau = \frac{2}{W_0 \left(\frac{(1-(2P_w)^{m+1})(1-P_w)+2^m \left(P_w^{m+1}-P_w^{R+1}\right)(1-2P_w)}{(1-2P_w)(1-P_w^{R+1})} \right) + 1} \tag{2.1}$$

where R is the number of the backoff stage (R = m+1 in above figure), W_0 is the minimum contention window size + 1, $m = log_2(\frac{CW_{max}}{CW_{min}})$ and each frame collides with constant collision probability p given by equation 2.2

$$P_w = 1 - (1 - \tau)^{n-1} \tag{2.2}$$

where n is the number of competing stations.

The two equation with two variables can be solved to obtain τ and p. Using obtained τ, the normalized throughput S can be calculated according to equations 2.3- 2.8, where T_s is the average time that the channel is sensed busy due to successful transmission, T_c due to collision, $E[P]$ is the frame time duration and P_{tr} is the probability of at least one station transmitting in a slot given by equation 2.8 and P_s is probability of successful transmission as in 2.8. Here H represents the MAC and PHY header time, and δ is the propagation delay equal to 0.1 us. We investigate basic access mechanisms, and below are the governing equations. Network throughput is then calculated as $(DataRate) * S$.

$$S = \frac{P_s P_{tr} E[P]}{(1 - P_{tr})\sigma + P_{tr} P_s T_s + P_{tr}(1 - P_s)T_c} \tag{2.3}$$

$$T_s = H + E[P] + SIFS + \delta + ACK + DIFS + \delta \tag{2.4}$$

$$T_{c,old} = H + E[P] + DIFS + \delta \tag{2.5}$$

In accordance with the current IEEE 802.11 standard specification, during contention, the node waits for an Ack time out period which leads to updating the collision time from $T_{c,old}$ defined in [1] to T_c [4] as

$$T_c = H + E[P] + SIFS + ACK + \delta + DIFS + \delta \tag{2.6}$$

$$P_{tr} = 1 - (1 - \tau)^n \tag{2.7}$$

$$P_s = \frac{n\tau(1 - \tau)^{(n-1)}}{P_{tr}} \tag{2.8}$$

Table 2.1 presents parameters employed for the three standards in ns-3 simulation and 2.2 shows the throughput for all MCSs of 802.11a calculated in MATLAB using an equation solver.

Table 2.1.: Parameters for simulation (802.11a)

Slot time σ (us)	9
SIFS (us)	16
DIFS (us)	34
PHY Header (us)	4us * 5 OFDM symbols
MAC Header (Bytes)	36
ACK (us)	PHY + 112/dratemin
UDP + IP Header (Bytes)	28
R	1000
CW_{min}	15
CW_{max}	1023
m	6

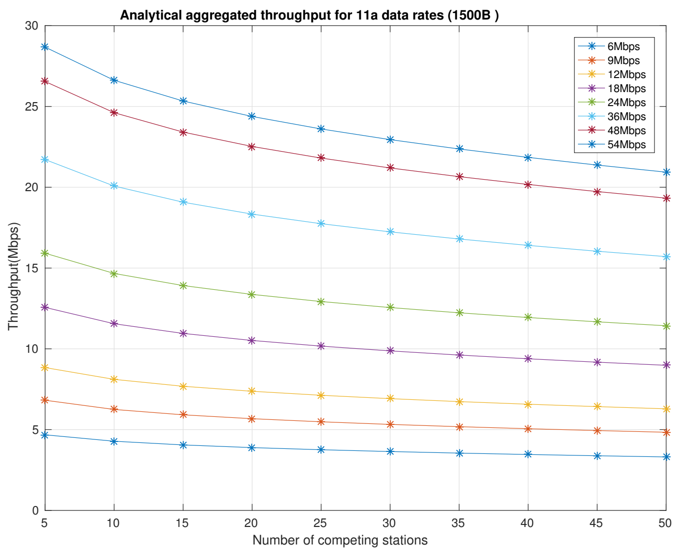

Figure 2.2.: Throughput performance from mathematical model

3. ns-3 Network Throughput Validation

Test Setup:

We have constructed ns-3 simulation scenarios to match the analytical model as closely as possible. We use an ad-hoc network with varying numbers of nodes; the topology is shown in Figure 3.1. Unlike infrastructure networks with access points, the additional channel activity due to association (beacon transmission, active scanning etc.) are avoided; thus we may obtain relatively closer throughput results as most mathematical models generally consider only payload packets and association effects are not considered.

For ns-3 simulations, the setup is configured for a number of competing stations in the range [5, 50]. We conducted 10 simulation trials (with differently seeded random number generators) with N (number of stations) as 5, 10, 15 ..., 50 for 6 Mbps data rate, 802.11a, and averaged the results.

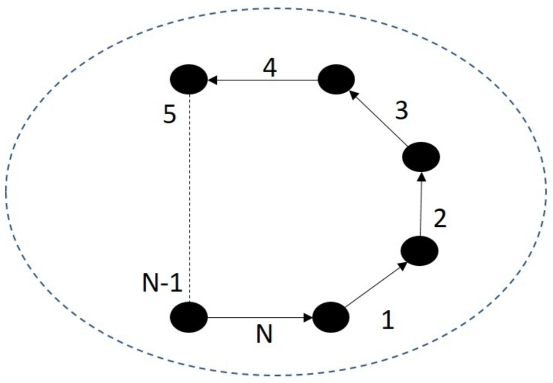

Figure 3.1.: Network Topology

The aggregated network throughput is calculated after the transition period because of steady state Markov modeling assumption in calculating the theoretical throughput.

ns-3 implementation:

In this experiment, we create "networksize" number of nodes in the network. The default YANS model for the physical layer is utilized, and the nodes are placed close to each other with a minimum distance of 0.001m.

The data packets are generated at the socket at an interval of 200 microseconds to make sure that saturation conditions exist at the 54 Mbps rate and below. No IP and transport layer headers are attached to the packet. Every node i transmits to node i+1, except that the last node transmits to the first node (a circular fashion as in Figure 3.1).

Below is the corresponding ns-3 script, written in C++.

```
int
Experiment::Run (const WifiHelper &wifi, const YansWifiPhyHelper &wifiPhy,
                 const WifiMacHelper &wifiMac, const YansWifiChannelHelper
                 &wifiChannel, uint32_t pktSize, uint32_t networkSize,
                 double delta, uint32_t gridWidth, double duration,
                 bool tracing)
{
  NodeContainer c;
  c.Create (networkSize);

  YansWifiPhyHelper phy = wifiPhy;
  phy.SetChannel (wifiChannel.Create ());

  WifiMacHelper mac = wifiMac;
  mac.SetType ("ns3::AdhocWifiMac");
  NetDeviceContainer devices = wifi.Install (phy, mac, c);

  MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                                  "MinX", DoubleValue (0.0),
                                  "MinY", DoubleValue (0.0),
                                  "DeltaX", DoubleValue (delta),
                                  "DeltaY", DoubleValue (delta),
                                  "GridWidth", UintegerValue (gridWidth),
                                 "LayoutType", StringValue ("RowFirst"));

  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (c);

  PacketSocketHelper packetSocket;
  packetSocket.Install (c);

  uint32_t nNodes = c.GetN ();

  for (uint32_t i = 0; i < nNodes; ++i)
    {
      uint32_t j = (i + 1) % nNodes;

      PacketSocketAddress socketAddr;
      socketAddr.SetSingleDevice (devices.Get (i)->GetIfIndex ());
      socketAddr.SetPhysicalAddress (devices.Get (j)->GetAddress ());
      socketAddr.SetProtocol (1);

      Ptr<PacketSocketClient> client = CreateObject<PacketSocketClient> ();
      client->SetRemote (socketAddr);
      c.Get (i)->AddApplication (client);
      client->SetAttribute ("PacketSize", UintegerValue (pktSize));
      client->SetAttribute ("MaxPackets", UintegerValue (0));
      client->SetAttribute ("Interval", TimeValue (MicroSeconds (200)));

      Ptr<PacketSocketServer> server = CreateObject<PacketSocketServer> ();
      server->SetLocal (socketAddr);
      c.Get (j)->AddApplication (server);
    }

    // Log packet receptions
    Config::Connect ("/NodeList/*/$ns3::Node/ApplicationList/*/
$ns3::PacketSocketServer/Rx", MakeCallback (&SocketRecvStats));


  Simulator::Schedule(Seconds(100.0),&restart_calc);
  Simulator::Stop (Seconds (duration));

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

This run method of class Experiment is called in the main() function below. We set the total duration of simulation to be 200 seconds, and later 100 seconds is considered for the throughput calculation.

In the script, we use YANS wifihelper for setting wifi channel configuration. The propagation delay is considered according to speed of light in a vacuum, and a log-distance path loss model is configured (but has little effect on the results due to the physical proximity of the nodes). To have the throughput vs network-size plot, we run for 5 to 50 nodes in the network with increment of 5 nodes. Further, we calculate the standard deviation on each throughput point in the curve as we run the simulation for 5 trials for each point.

```
int main (int argc, char *argv[])
{

  double duration = 200;
  uint32_t netSize = 50;
  uint32_t pktSize = 1500;
  double delta = 0.001;
  uint32_t trials = 3;
  uint32_t gridWidth = 10;
  double P_col=0;

  Config::SetDefault ("ns3::WifiRemoteStationManager::MaxSlrc",
  UintegerValue (10000));

  // Align with OFDM standard values
  Config::SetDefault ("ns3::DcaTxop::MinCw", UintegerValue (15));
  Config::SetDefault ("ns3::DcaTxop::MaxCw", UintegerValue (1023));


  std::stringstream ss;
```

```
ss << "wifi-11a-" << netSize << "-p-" << pktSize << "-throughput.plt";
std::ofstream netSizeThroughputPlot (ss.str ().c_str ());
ss.str ("");
ss << "wifi-11a-" << netSize << "-p-" << pktSize << "-throughput.eps";

Gnuplot gnuplot = Gnuplot (ss.str ());

WifiHelper wifi;
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);
WifiMacHelper wifiMac;
wifiMac.SetType ("ns3::AdhocWifiMac");
YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
YansWifiChannelHelper wifiChannel;
wifiChannel.SetPropagationDelay ("ns3::ConstantSpeedPropagationDelayModel");
wifiChannel.AddPropagationLoss ("ns3::LogDistancePropagationLossModel");

NS_LOG_DEBUG ("6");
Experiment experiment;
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                              "DataMode", StringValue ("OfdmRate6Mbps"));

Gnuplot2dDataset dataset;
Gnuplot2dDataset dataset_Model;
dataset.SetErrorBars (Gnuplot2dDataset::Y);
dataset.SetStyle (Gnuplot2dDataset::LINES_POINTS);
dataset_bianchi.SetStyle (Gnuplot2dDataset::LINES_POINTS);

double mean_t, throughput, stDev, throughputVector[trials];

for (uint32_t n = 5; n <= netSize; n += 5)
  {

    mean_t = 0;

    for (uint32_t run_index = 1; run_index <= trials; run_index++)
      {
        std::fill(bytesReceived.begin(), bytesReceived.end(), 0);
        std::fill(packetsReceived.begin(), packetsReceived.end(), 0);
        throughput = 0;


        experiment.Run (wifi, wifiPhy, wifiMac, wifiChannel, pktSize,
        n, delta,
        gridWidth, duration, false);

        for (uint32_t k = 0; k< n; k++)
          {
            throughputpernode[k] = (double)bytesReceived[k]/1000/1000/100*8;

            throughput += throughputpernode[k];
          }
        std::cout << "Total␣throughput␣" << throughput << std::endl;
        mean_t += throughput;
        throughputVector[run_index - 1] = throughput;
      }
    mean_t = mean_t / trials;
    stDev = 0;
    for (uint32_t i = 0; i < trials; ++i)
      {
        stDev += pow (throughputVector[i] - mean_t, 2);
      }

    stDev = sqrt (stDev / (trials - 1));
    dataset.Add (n, mean_t, stDev);
    std::cout << mean_t;
  }
```

4. Discussion

All simulations in the section are conducted for a duration of 200 seconds. To obtain the best results, the initial transient time should be avoided in the throughput calculation. We observed throughput fluctuation for initial 20 seconds in 50 node simulation (12Mbps) shown in Figure 4.1. The transient time for higher rate would be higher, so throughput is calculated 100 seconds after simulation start time. We obtain close correspondence between results for ns-3 simulation and those from the mathematical model as shown in Figure 4.2. Figure 4.3 captures the effect of packet size on throughput.

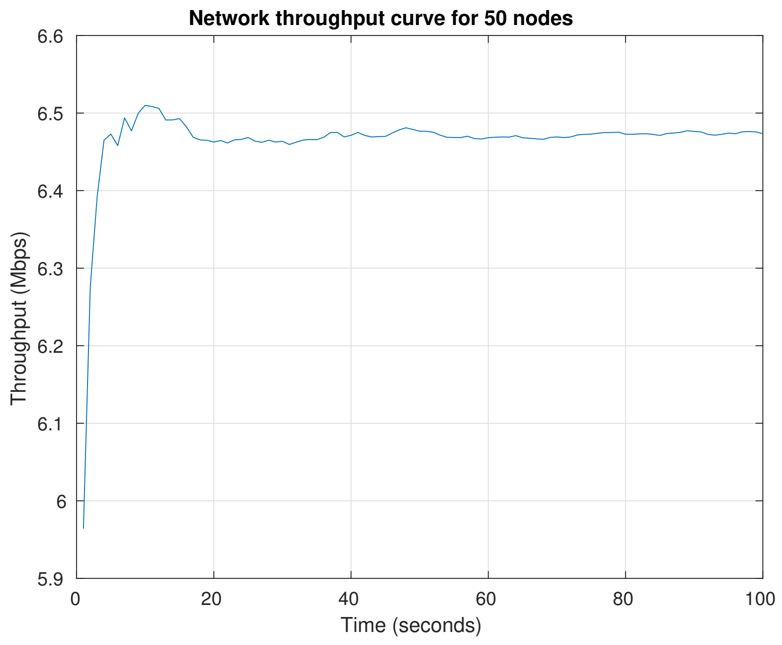

Figure 4.1.: Network throughput fluctuation with time for 50 nodes simulation in 1 trial for 12Mbps.

In 802.11a standard, the maximum number of retry limit is chosen to be 7. The mathematical model and ns-3 simulation results for throughput are compared in figure 4.4.

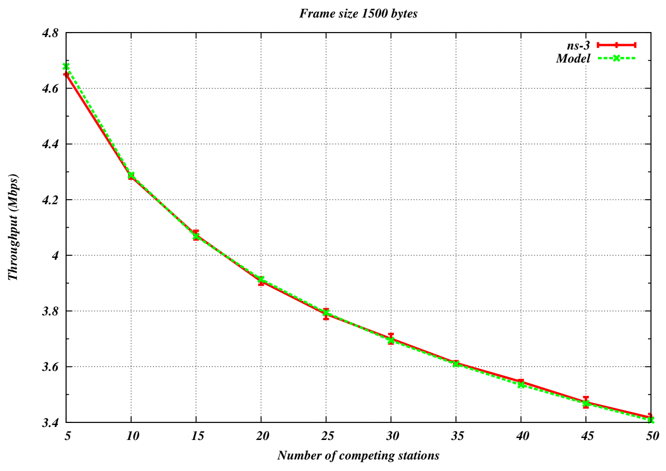

Figure 4.2.: Throughput validation results for 11a 6 Mbps for 5 runs

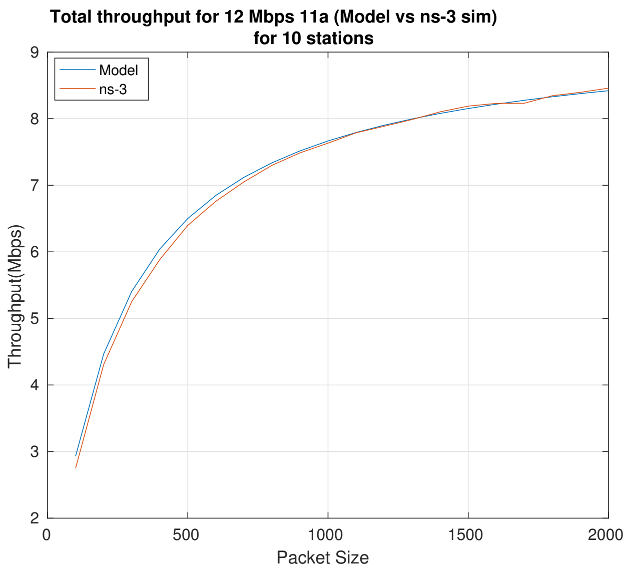

Figure 4.3.: Network throughput for 12 Mbps data rate for 10 stations wrt packet size

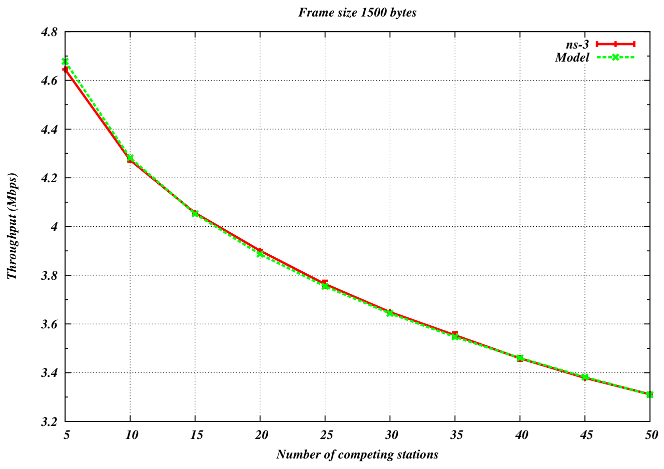

Figure 4.4.: Throughput validation results for 11a 6 Mbps for 5 runs

5. Summary

In this work, we have validated the ns-3 MAC layer implementation configured for the IEEE 802.11a standard against well known bi-dimensional Markovian mathematical models for aggregated network saturation throughput under ideal channel conditions. Comparison of simulation results with the mathematical model shows close correspondence.

We observed that the performance of the basic access method strongly depends on the DCF-related system parameters, mainly minimum contention window and number of stations in the wireless network, but less so on other details of the standard. Hence, this model can be used for validation of similar Wi-Fi standards such as 11n/11ac/11ax.

Furthermore, with the acceptance of LTE in unlicensed bands, we suggest as future work to study the performance of the DCF against LTE unlicensed systems, including the development of analytical models and corresponding simulations.

Bibliography

[1] G. Bianchi. Performance analysis of the ieee 802.11 distributed coordination function. *IEEE Journal on selected areas in communications*, 18(3):535–547, 2000.

[2] M. Morteza. Analytical modeling of wi-fi lte-laa coexistence: Throughput and impact of energy detection threshold. *IEEE Transactions on Networking, submitted*, 2017.

[3] G. Pei, T. Goff, and T. Henderson. Validation of ns-3 wifi distributed coordination function (dcf) model. 2012.

[4] I. Tinnirello, G. Bianchi, and Y. Xiao. Refinements on ieee 802.11 distributed coordination function modeling approaches. *IEEE Transactions on Vehicular Technology*, 59(3): 1055–1067, 2010.

A. Appendix

To analyze the ns-3 experimental throughput behaviour at higher data rates, we run the above network topology for data rate of 9, 12, 18, 24, 36, 48 and 54 Mbps, throughput is as shown in Figure A.1 to A.7 respectively.

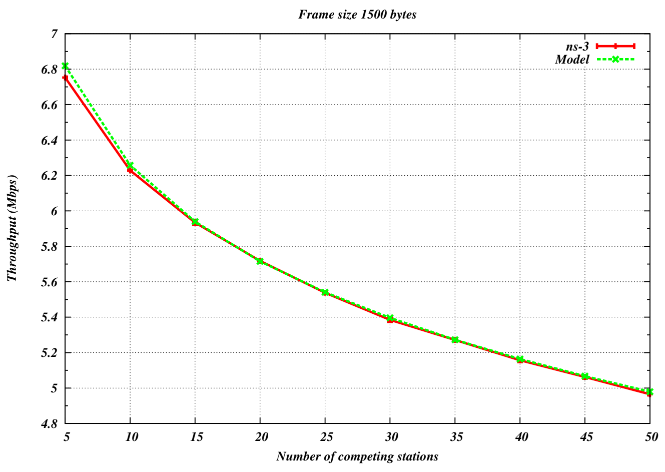

Figure A.1.: Throughput validation results for 9 Mbps for 200 seconds and 5 runs

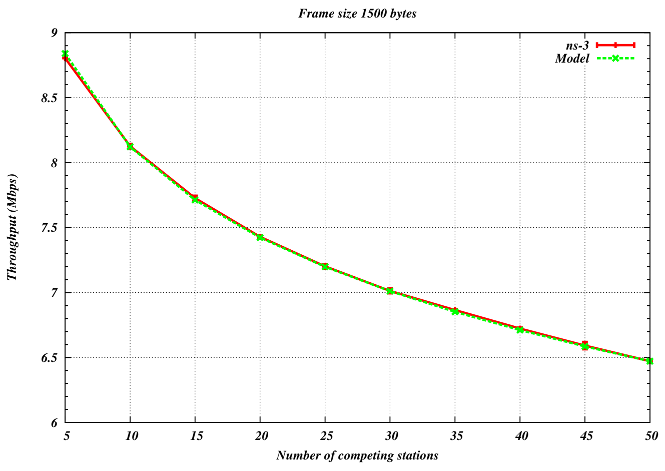

Figure A.2.: Throughput validation results for 12 Mbps for 200 seconds and 5 runs

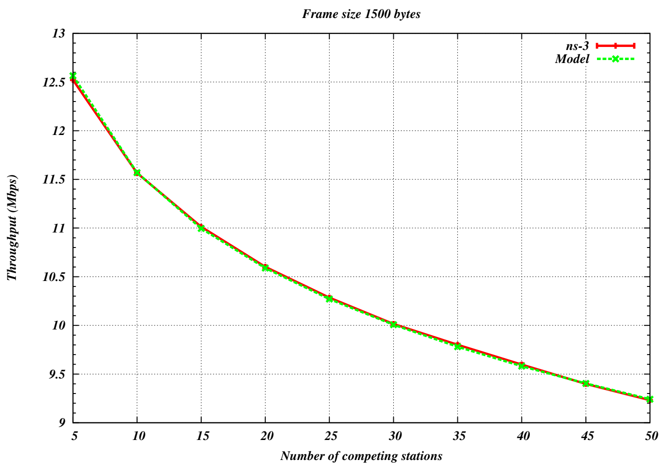

Figure A.3.: Throughput validation results for 18 Mbps for 200 seconds and 5 runs

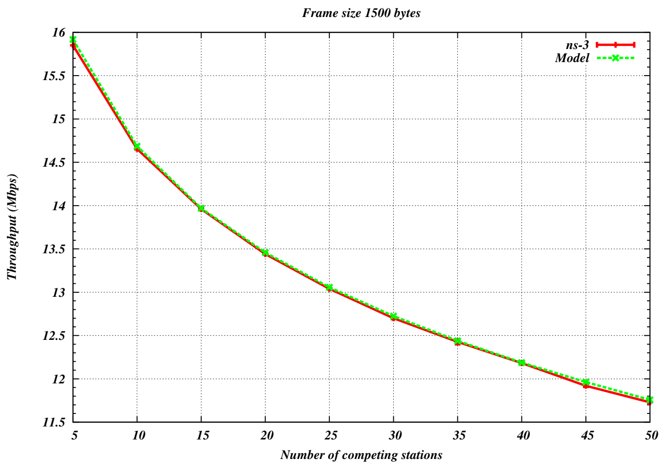

Figure A.4.: Throughput validation results for 24 Mbps for 200 seconds and 5 runs

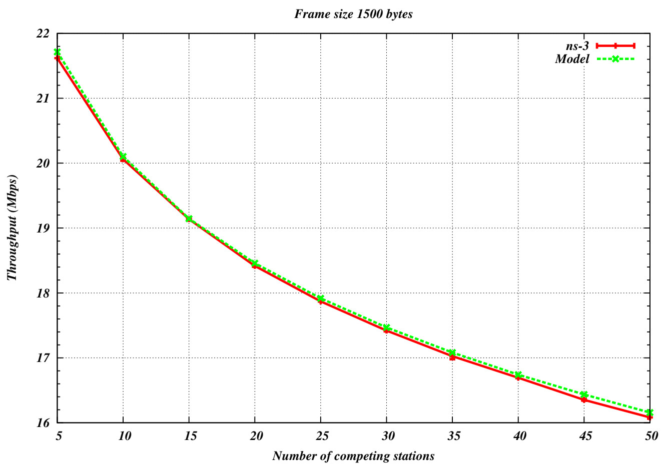

Figure A.5.: Throughput validation results for 36 Mbps for 200 seconds and 5 runs

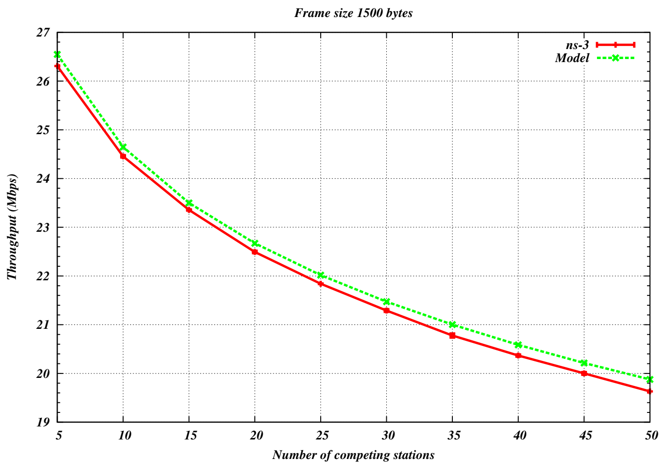

Figure A.6.: Throughput validation results for 48 Mbps for 200 seconds and 5 runs

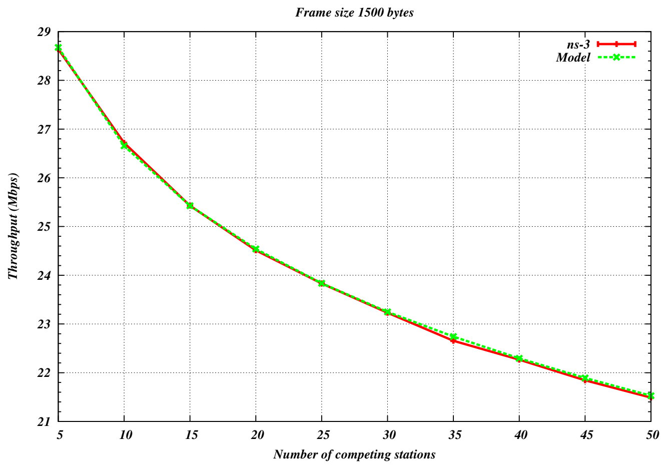

Figure A.7.: Throughput validation results for 54 Mbps for 200 seconds and 5 runs