Network Simulator-2 modifications for 802.11b with Auto Rate Fallback

Kevin Springborn

1 Introduction

The Network Simulator 2 by default uses a Black and White model for packet reception, where two nodes are either in range (every bit is correctly received) or out of range (all bits are lost). Three components are needed to accurately model modern wireless systems: 1) A signal to noise based bit reception model 2) A data rate selection algorithm 3) Accurate modeling of the 11 802.11b channels. Contemporary wireless systems change the data transmission rate depending on the clarity of the communication. A high rate selection can transmit data almost 11 times faster than the slowest data rate. Rate selection algorithms are an important key to modeling wireless systems correctly. Unfortunately the Black and White model does not allow for realistic rate selection. Two nodes in range will eventually switch to the highest rate; lower rates will never be used. Modeling rate selection requires a grey packet reception model, where each bit has a probability of correct reception. Finally, 802.11b has three orthogonal channels, on which data can be transmitted concurrently. 802.11b channel 3 under the right conditions can receive traffic transmitted on channel 1. This cross channel interference must be incorporated into a simulation of a dense wireless deployment. This work presents extensions to ns-2 to allow for more realistic simulations of wireless communication.

The extensions were tested in a wireless only scenario, using some of the Monarch logging functionality. While the author suspects the code may work in more exotic situations, such as wired-cum-wireless or satellite scenarios, the code has not yet been tested.

This document is designed to help an experienced ns-2 programmer understand the changes in the extensions and also to help the user understand the current limitations of the extensions.

2 Signal to Noise Ratio(SNR) Model

2.1 TCL Environment Variables

- Phy/WirelessPhy set ambient_noise_ *double*
  - Sets the background noise used for all SNR calculations
- Phy/WirelessPhy set Pt_ *double*
  - Sets the transmission power of wireless nodes
- Node/MobileNode set error_ *double*
  - Sets a fixed chance of packet error for all nodes in the system. Irrespective of SNR calculations
- Phy/WirelessPhy set CPThresh_ *double*
  - This value has become the minimum ratio of Rx powers needed in order to capture the channel

2.2 TCL Commands

- $*mobilenode* change-error *double*
  - Modifies the fixed chance of packet error for a specific node

2.3 Implementation

A special error model is inserted in the node on the inbound pathway. The wireless-phy tracks noise through the use of a noise element list. A packet that is not received or carrier sensed adds an entry into the noise list. The entry consists of the reception power and the duration of the reception. This allows the wireless-phy to attach an estimate of the current noise at the start of the reception to incoming packets by storing the value in a field added to the packet-stamp. The packet-stamp is information not present in the simulation, but is information attached to the packet for use by the simulator. The packet-stamp for each received packet is modified to record the noise at time of reception. The error model uses the reception noise along with the reception power (also stored in the packet-stamp) to calculate the probability of reception.

A packet consists of two parts a preamble and a main section. The preamble must be transmitted at the lowest data rate (1mbps), while the main data can be transmitted at higher rates. The error model calculates bit error probabilities for each section independently and multiplies bit error probabilities to get a packet reception probability. The data rate is not stored in the packet. It is calculated based on the size of the packet, size of the preamble, and the time taken to receive that packet. The probability of receiving a bit is 1- (erfc(sqrt((Rx_power * Bandwidth)/(noise * bitRate)) / 2). The bandwidth of 802.11b is 2MHz.

The data structure for managing the noise is a sorted linked list, with each element containing the noise value and its expiration time. The noise elements are sorted by increasing expiration time, so the element expiring first is at the head of the list. At any time the noise at a node is the sum of the noise from all its current noise elements. On every read expired noise elements are removed from the list.

2.4 Notes

The SNR simulation extension is greatly dependent on node placement. For transmission ranges of about 250m the absolute antenna height should be .9 m above the ground (note antenna height is the node position + the antenna position relative to the node)

2.5 Inaccuracies

The major flaw still present in this calculation is that the reception probability is based on the noise at the instant the packet begins arriving at the host. In reality a burst of noise arriving while the packet is being received will cause bit errors, but in the current implementation the burst will have no effect. The solution to this is based on the fact that only one packet can be received at a time. Upon reception start a linked list of during_reception_noise should be initialized. Every packet reception while the wireless-phy is in the receive state should be added to the during_reception_noise list. The error model can then use this list (can be access since the error model's downtarget is the mac, which's down target is the wireless-phy) to correctly calculate reception.

Also the Complementary Code Keying(CCK) used at the 5.5MHz and 11MHz levels of 802.11b is not correctly modeled. At the 5.5MHz and 11MHz levels the error model is only an approximation. To fix this actual keys would need to be implemented and the reception probability for each key calculated. Notice that some codes tolerate corrupted keys.

Currently the carrier sense threshold is set to the background noise level. If the reception power is less than the ambient_noise, then the packet does not carrier sense. Look in wireless-phy::sendUp() to change this behavior.

3 Auto Rate Fallback(ARF)

3.1 TCL Environment variables

- Mac/802_11 set Arf_ *(1 or 0)*
  - A value of 1 enables the default ARF algorithm for all nodes
  - A value of 0 disables ARF for all nodes

3.2 TCL Commands

- $*mobilenode* enable-arf

  - Enables ARF for a specific node

- $*mobilenode* disable-arf
  - Disables ARF for a specific node

3.3 Implementation

The current ARF algorithm uses a simple count of consecutive receptions and failures, along with a probation period. Receptions and failures are the result of individual transmissions, such that a packet retransmitted once would count as a failure followed by a reception. The constant values can be modified in arf.h. 4 consecutive receptions trigger a rate increase if possible, while 2 consecutive failures trigger a rate decrease. Upon moving to a higher transmission rate a probation period of 1 packet is used. If there are any failures during the probation period the rate is immediately decremented.

All old references to the dataRate_ value were replaced with a call to getDataRate(). The getDateRate() function checks to see if an ARF object is defined. If the ARF object is defined, then getDataRate() returns the rate specified by the ARF object. If the ARF object is not defined, then getDataRate() returns the value in dataRate_.

Two modifications are needed to inform the ARF object of receptions and failures. A successful transmission is signaled by the correct reception of an ACK from the destination. The successfulTransmission() function of the ARF object is called in recvACK() in the mac-802_11 code. A failed transmission is signaled by the retransmission of the data packet. The failedTransmission() function of the ARF object is called in RetransmitDATA() in the mac-802_11 code.

3.4 Inaccuracies

None known.

4 Channel Switching

4.1 TCL Environment variables

- Node/MobileNode set switch_time_ *double*
  - Sets the number of seconds required for a node to change its interface from one channel to the next. Note: packets transmitted during switch are delayed, while packets received are dropped.
- Node/MobileNode set clear_rt_on_switch *int*
  - This currently only applies to AODV routing. A value of 1 causes the routing tables to be cleared upon switching channels. While a value of 0 leaves the routing tables in tact.

4.2 TCL Commands

- $*mobilenode* switch-channel *double*
  - Switches a node's wireless interface to the channel specified. During the switch packets may be lost or delayed. (New nodes remain on channel 1 until changed)

4.3 Implementation

The basis of channel extension is adding channel information to the packet-stamp and using that information in propagation to correctly calculate the received power. The nodes current channel (channel_number_) is stored in the mobilenode object. Before the wireless-phy sends the packet onto the channel it modifies the packet-stamp to include the node's current channel. Tworayground::Pr() compares the packet stamps and modifies the transmission power passed to the propagation equations. The difference between the channel numbers is used to lookup the decrease in power.

Interface channel switching delays are accomplished by storing the "interface up time" in the mac and checking this in mac-802_11::recv() and mac-802_11::send(). When a node switches channels value of (current_time + interface_switching_delay) is stored in mac::interfaceUpTime. mac-802_11::recv() compares the current simulation time with the time stored in mac::interfaceUpTime. If the interface is switching any packets traversing the receive pathway are dropped. mac-802_11::send() compares the current simulation time with the time stored in mac::interfaceUpTime. If the interface is switching channels the outgoing packet is stored and a timer is set for the time the interface comes back up. When the timer is fired the packet continues along the send pathway.

The switchChannel function in mobilenode allows the option of clearing AODV routing tables when a node is switched. switchChannel() simply calls a method to clear the routing tables from the AODV object.

4.4 Inaccuracies

Switching a channel takes place as soon as the command is issued. In reality an interface could not switch channels until its current transmission or reception is complete. The effect is that the ns-2 extension models quicker switching time that specified in the case a reception or transmission was taking place when the interface began changing channels.