An Investigation of Policies for Caching Radio Propagation Calculations in the Simulation of Wireless Networks

Progress Report

Michael Dippery
mdippery@bucknell.edu
Luiz Felipe Perrone (Faculty Mentor)

September 1, 2007

This past summer, I worked with Professor Perrone on his wireless networking simulator, SWAN. The goal of my research was to investigate caching policies for the radio propagation calculations in SWAN, with the hopes of making the simulator operate more efficiently. My initial work grafted a cache onto the the simulator. The cache stored frequently used computations for quick lookup and retrieval.

Unfortunately, early tests showed that, while the cache operated as designed, it did *not* increase the efficiency of SWAN, and in some cases even degraded it. Analysis of test data showed that the speed gained in looking up recently-used calculations was lost in the actual lookup and retrieval of those calculations.

Fortunately, there were other avenues through which we could approach the caching problem. Schmitz and Wenig described a method of calculating radio propagation by using ray-tracing techniques commonly found in computer graphics [4]. This technique measures the radio wave propagation from a point T_x to another point R_x. These measurements are collected in a file, which can then be retained between simulation runs in files referred to as "propagation maps".

To enable this functionality, Schmitz and Wenig used a data structure known as a k-dimensional binary search tree, or k-d tree for short [4]. This data structure was first described by Jon Louis Bentley [1]. The primary benefit of using a k-d tree is that it has excellent support for finding *nearest neighbors*; given an arbitrary point, the k-d tree is uniquely suited to finding the nearest data points to that arbitrary point.

I contacted Schmitz and Wenig in an attempt to obtain some sample ray-tracing data, as well as their implementation of the k-d tree to use as a reference. Unfortunately, they were uncooperative and uninterested in a sharing of information, which forced me to implement the k-d tree myself. This required reading Bentley's original paper on the subject, as well as more papers by Bentley about the data structure and nearest neighbor searches [3, 2]. The C++ implementation of the k-d tree for SWAN took a few weeks, but I completed that work by the end of July.

I then had to design a file format for the propagation maps. I decided to use XML as the file format, since XML is useful in describing hierarchical data, and can be parsed using open-source parsers. I then developed a schema that ensures that XML-based propagation map files are well-formed.

My next task was to integrate an XML parser into SWAN. I used a C-based parser called Expat. Expat is released under the free-software MIT license, and thus could be used in SWAN. Thanks to Expat, I did not have to write any XML parsing software from scratch, and thus got this feature running in SWAN in short order. Integrating Expat simply required me to include the source code in SWAN's, and alter SWAN's Makefile to build and link the Expat static library. Furthermore, the integration of an XML parser into SWAN not only enables the use of propagation maps, but may have benefits for other parts of the simulator down the road, if other data is defined using XML.

My final task was to assess the performance of my new module that utilizes the ray-traced propagation map, in order to assess whether or not the efficiency of the module could be improved with another layer of caching. To assess the performance, I used a tool from Intel called "VTune" to track in which function SWAN spent most of its execution time. I obtained the shown below:

Method	**Runtime (%)**
`KDTree<T>::insertNode()`	≈21%
`KDTree<T>::findNearest()`	<2%

The first method, `insertNode()`, is used when building the tree from the XML file, and the second, `findNearest()`, is used when retrieving data about the propagation map from the k-d tree. The recovery of data from the tree is thus fairly efficient, and most likely, a cache would not be extremely beneficial in the use of the propagation map. However, the propagation map module itself is quite useful since it allows a map to be created before execution time and saved for future runs, and the data structure used to model the map (viz. the k-d tree) is quite efficient in its retrieval mechanism.

References

[1] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[2] Jon Louis Bentley. K-d Trees for Semidynamic Point Sets. Technical report, Association for Computing Machinery (ACM), 1990.

[3] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[4] Arne Schmitz and Martin Wenig. The Effect of the Radio Wave Propagation Model in Mobile Ad Hoc Networks. In *MSWiM '06*, pages 61–67. Association for Computing Machinery (ACM), October 2006.