Minimizing Wireless Traffic Interference for User-Rich LANs With Dynamically Reconfigured Hierarchical Network Topologies

Erik Gabrielsen, Ian Johnson, Danh Nguyen, Gavin Pham, Alex Saladna, Travis Siems
{egabrielsen, ianj, danhn, gpham, asaladna, tsiems}@smu.edu

ABSTRACT

Erik Got dis

1. INTRODUCTION

Local Area Networks (LANs) which operate over a wireless medium using variants of the 802.11 protocol provide remarkable utility to users by allowing them to connect and disconnect from a network at any time without forming their own ad-hoc network to communicate with nearby users. To achieve such functionality, LANs typically operate by connecting each new user to a statically assigned physical access point (AP) which routes the user's traffic to the rest of the network. LANs often consist of tens or hundreds of users who share the wireless link among themselves using a variety of Media Access Control (MAC) algorithms and bandwidth distribution strategies such as Frequency Division Multiplexing (FDM) and Code Division Multiple Access (CDMA). While these approaches efficiently distribute network resources among nodes in a low or medium traffic density network, they deteriorate rapidly in high host/traffic density (user-rich) LANS. In all existing MAC algorithms, successful frame throughput declines significantly as host and traffic density increases on the network. Likewise, static multiplexing strategies such as FDMA and CDMA have performance issues in high-density wireless networks.

There is, however, a third viable approach to throughput optimization for user-rich LANs. Breaking down a large LAN into sub-LANs which operate within their own geographic region can significantly improve the per-host efficiency of the overall LAN. This is often done in practice using multiple physical APs within the geographic span of a LAN. However, this approach statically allocates network resources, which causes it to suffer the same fate as FDMA when uneven network activity distributions occur. A dynamic approach with multiple physical APs is not realistic, however, as it would require that APs physically move around the network in response to varying network activity. Therefore, our research explores an algorithmic approach to dynamic subnetwork generation in which the physical root AP dynamically assigns network users to act as subAPs which route traffic from nearby users to the AP.

In organizing a set of subnetworks, a number of considerations must be taken into account. If an algorithm is to be considered superior to the typical existing star topology, in which all hosts connect directly to a physical AP, it must outperform the star topology in most, if not all network viability metrics for a wide variety of possible network distributions. A typical LAN may be geographically organized in a normal or pseudo-random distribution, where hosts have no tendency to gravitate toward one another, or they may exist in a clustered distribution wherein hosts tend to group together. To entertain the idea of a clustered distribution, consider two scenarios. The first is a static clustering distribution, where users gravitate around a physical landmark within the geographic span of a LAN. An example of such a scenario would be an area of seats at a gate at an airport. A LAN operating in an airport is likely a user-rich LAN, and users tend to cluster together around seating areas which do not change over time. A second scenario is a dynamically clustered distribution, where the geographic location of a cluster is subject to change over time. An example would be a large conference room, wherein people move around in groups while each of their phones continuously talk to an AP. The two scenarios will be treated as one agglomerate 'clustered' distribution for the purposes of our research.

After exploring existing approaches to throughput optimization via multiplexing, MAC algorithms, and network topology optimization, we will define our model for assessing topology viability and explore the behaviour of a number of algorithms used to organize a hierarchical network topology.

Related Work Cite and explain at least one paper about multiplexing strategies (e.g. FDM)

Cite and explain at least one paper about a MAC algorithm (e.g. the Aloha paper)

Cite and explain at least one paper about a LAN topology algorithm. We'll implement this/these algorithm and compare it to our results. (e.g. Prim's algorithm)

Cite AT LEAST 5 works.

2. TOPOLOGY VIABILITY MODEL

In order to ascertain the viability of a LAN topology, our model considers a set of possible geographic host distributions ranging from randomized distributions to heavily clustered distributions, as well as a variety of average host densities per geographic area which have been discretized into three categories: low, medium, and high-density. While our focus lies in the performance of LAN topology algorithms in high-density (user-rich) LANs, a viable algorithm must also provide competitive functionality compared to other algorithms in low and medium-density host distributions.

Our model operates under a set of assumptions which streamline the process of assessing the viability of topology algorithms.

- **Constant transmission rates:** All hosts send data across the network at a constant rate. In practice, this rate would be the average data transmission rate of a host.
- **Single access point:** All hosts connect, either directly or indirectly, to a single root AP. Because each algorithm tested (excluding the null algorithm, the star topology) generates a hierarchical topology, it is assumed that each algorithm could be reasonably extended to a multi-AP system.
- **Mutable NIC broadcast range:** All NICs can change the power/range with which they transmit a broadcast signal. The lack of this assumption precludes any significant interference minimization.
- **Hosts within range of AP:** All hosts on the LAN are within range of the AP such that each host can broadcast a signal with sufficient strength to reach the AP and the AP, in turn, can broadcast a signal with sufficient strength to reach each host.

Provided this set of assumptions, our model measures the viability of an algorithm for any given environment based on a number of topology performance metrics.

- **Interfering node count:** The number of nodes whose broadcasts interfere with the communications of any given node.
- **Number of re-broadcasts:** The number of nodes through which the communications of any given node are routed before reaching the AP.
- **Signal distance to AP:** The total distance over which signals from a host travel before eventually reaching the AP.
- **Total network traffic:** The total amount of communication occurring over the network at any given time. Calculated as the sum of the transmission rate of each node multiplied by the number of re-broadcasts that must occur to connect a host to the AP.

Each of these metrics will be calculated for each proposed algorithm in many possible host layout scenarios.

As a basis for comparison of topology algorithms, a star topology algorithm will be used as a null algorithm. In a star topology, each host on the network connects directly to the AP. For each topology algorithm tested, the algorithm will be used to generate a topology for 50 networks from each possible combination of semi-clustered, very-clustered, and unclustered (random) host distributions with low, medium, and high host densities.

Figure 1 shows the format in which topology viability data will be presented. The 'Layout' column refers to the geographic distribution of hosts in the form $Layout : Density$, where $layout$ is R for random, SC for semi-clustered and C for clustered, and $density$ describes the geographic host density as L for low, M for medium, and H for high-density. The interference, hops, distance, and traffic columns refer to the four performance metrics described above (interfering node count, number of rebroadcasts, signal distance to AP, and total network traffic, respectively).

Layout	Interference	Hops	Distance	Traffic
R:L	8.93 ± 4.39	0 ± 0	272 ± 117	168
R:M	29.6 ± 11.3	0 ± 0	307 ± 115	508
R:H	72.4 ± 27.2	0 ± 0	307 ± 118	1208
SC:L	16.3 ± 5.32	0 ± 0	240 ± 126	142
SC:M	32.4 ± 11.7	0 ± 0	315 ± 102	450
SC:H	74.9 ± 27.8	0 ± 0	287 ± 103	1280
C:L	17.7 ± 5.69	0 ± 0	307 ± 140	127
C:M	40.7 ± 15.9	0 ± 0	289 ± 101	528
C:H	71.2 ± 30.9	0 ± 0	346 ± 144	1252

Figure 1: Viability metrics for the null algorithm

3. PROXIMITY GREEDY ALGORITHM

Our first approach to generating an ideal network topology is to prioritize nodes geographically proximate to the AP to become sub-APs. To do so, a greedy algorithm was implemented which assigns the most proximate unassigned host to become a sub-AP, and assigns the n nodes closest to the new sub-AP to be its children, where n is a load factor calculated as the total number of hosts on the network divided by the desired number of sub-APs. This approach attempts an even distribution of re-transmission loads among the hosts assigned to be sub-APs.

3.1 Proximity Greedy Algorithm

The most basic approach to topology optimization is with an un-optimized, single-iteration of the proximity greedy algorithm. This basic proximity algorithm, when tested against a set of geographic host layouts generated according to the specifications layed out in the previous section, generated network topologies whose relevant metrics are shown in figure 2.

Layout	Interference	Hops	Distance	Traffic
R:L	4.73 ± 3.27	0.76 ± 0.42	381 ± 112	238
R:M	9.34 ± 9.34	0.81 ± 0.39	350 ± 149	893
R:H	15.9 ± 22.2	0.86 ± 0.35	332 ± 128	2300
SC:L	6.11 ± 4.77	0.77 ± 0.42	316 ± 166	281
SC:M	10.9 ± 10.9	0.82 ± 0.38	361 ± 150	879
SC:H	19.1 ± 27.8	0.87 ± 0.34	334 ± 122	2548
C:L	6.20 ± 4.55	0.80 ± 0.40	269 ± 138	294
C:M	10.7 ± 10.8	0.86 ± 0.35	244 ± 100	869
C:H	20.7 ± 28.4	0.86 ± 0.35	326 ± 114	2248

Figure 2: Viability metrics for the basic proximity-greedy algorithm

3.2 Post-Configuration Optimization

Post-network configuration, the greedy algorithm can be optimized by having each non-subAP host switch to a new subAP if it is closer to a subAP other than its own. This slightly deteriorates the even distribution of traffic routed through each subAP which exists in the basic proximity greedy algorithm. Results for the optimized version of the proximity algorithm are shown in figure 3, and pseudocode for the optimized algorithm is shown below.

Algorithm 1 Optimized Proximity Greedy Algorithm

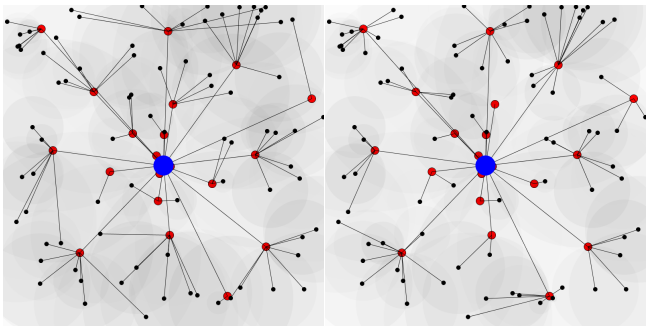
Require: *hosts* is a list of all hosts on the network, *subAPs* is an empty list of subAPs, and *rootAP* is the physical AP of the network

procedure BUILDNETWORK(*hosts*, *subAPs*, *rootAP*)
  sort hosts by distance to rootAP
  for $i \leftarrow 1$ to k **do**
    $subAPs[i] \leftarrow host[i]$
  for $i \leftarrow k$ to $numHosts$ **do**
    sort *subAPs* by distance to $host[i]$
    $host[i].myAP \leftarrow subAPs[0]$

Layout	Interference	Hops	Distance	Traffic
R:L	4.1 ± 3.10	0.67 ± 0.47	330 ± 117	266
R:M	7.61 ± 9.29	0.82 ± 0.38	335 ± 134	997
R:H	13.2 ± 24.4	0.86 ± 0.35	337 ± 117	2309
SC:L	6.03 ± 4.96	0.80 ± 0.40	312 ± 91	280
SC:M	7.73 ± 10.7	0.83 ± 0.38	327 ± 156	937
SC:H	15.0 ± 29.4	0.86 ± 0.34	339 ± 123	2193
C:L	7.23 ± 7.65	0.80 ± 0.40	307 ± 121	282
C:M	9.76 ± 14.7	0.84 ± 0.37	309 ± 131	999
C:H	13.5 ± 25.4	0.86 ± 0.35	243 ± 116	2302

Figure 3: Viability metrics for the optimized proximity-greedy algorithm

Figure 4 shows a comparison of the two algorithms for a randomly distributed data set. The grayed-out area represents geographic regions where interference is occurring. Darker shades of gray indicate greater interference.



(a) Un-Optimized (b) Optimized

Figure 4: Optimized vs Un-optimized

4. RECURSIVE PROXIMITY GREEDY ALGORITHM

As an expansion of the basic proximity greedy algorithm, two recursive versions exist which, after assigning sub-APs, allow each sub-AP to use the same proximity greedy algorithm used by the AP to assign their own sub-subAPs. Both a semi-recursive version, which restricts the network to generating a limited number of layers to the AP hierarchy, and a fully recursive version, which generates a tree of sub-APs until it reaches edge nodes on the LAN, are implemented.

4.1 Semi-Recursive Proximity Greedy Algorithm

In a semi-recursive approach, the proximity greedy algorithm recurses up to k times to define subnetworks, sub-subnetworks, and so on. k is calculated as a function of the total number of nodes on the network, such that the algorithm will recurse with depth relative to the density of the network. A viable, but less robust solution would be to simply recurse n times where n is

some constant hard-coded in the algorithm. This approach, however, fails to properly scale to total number of hosts on the network. Viability metrics for the network topologies generated by this algorithm in our test environment are shown in figure 4.1.

FIGURE 4.1 – TABLE OF STATS FOR SEMI RECURSIVE (NOT YET IMPLEMENTED ALGORITHM)

4.2 Fully-Recursive Proximity Greedy Algorithm

In a fully-recursive greedy approach, the proximity algorithm recurses indefinitely until it reaches edge nodes on each branch of recursion. This approach attempts to minimize total interference, not taking into account additional overhead introduced by having numerous re-broadcasts. The relevant metrics for a recursive topology algorithm are shown in figure 4.2, and pseudocode for the fully recursive algorithm is included below.

FIGURE 4.2 – TABLE OF STATS FOR FULLY RECURSIVE (ERIK ALGORITHM)

Algorithm 2 Recursive Proximity Greedy Algorithm

Require: *hosts* is a list of all hosts on the network, *subAPs* is an empty list of subAPs, and *rootAP* is the physical AP of the network

procedure BUILDNETWORK(*hosts*, *subAPs*, *rootAP*)
  if *length*(*hosts*) <= 1 **then**
    return
  sort hosts by distance to rootAP
  for $i \leftarrow 1$ to k **do**
    $subAPs[i] \leftarrow hosts[i]$
  for $i \leftarrow k$ to *numHosts* **do**
    sort *subAPs* by distance to $hosts[i]$
    $hosts[i].myAP \leftarrow subAPs[0]$
    $subAPs[0].myChildren.append(hosts[i])$
  for *subAP* in *subAPs* **do**
    buildNetwork(*subAP.myChildren*, [], *subAP*)

Figure N shows a topology generated by a fully-recursive distance greedy algorithm. The resulting network has very low interference, but a high number of re-broadcasts must occur to transfer information from edge nodes to the AP.

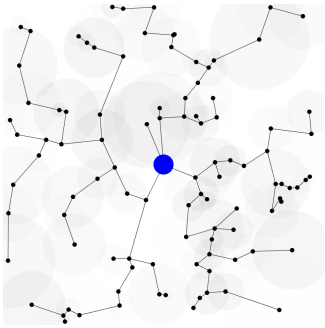


Figure 5: Optimized vs Un-optimized

5. CENTRALIZATION GREEDY ALGORITHM

In order to optimize for areas of high host density which are isolate from the location of the AP, a greedy algorithm is implemented which selects sub-APs based on hosts' centralization relative to all other hosts on the network, where centralization is defined as the average distance to each other host on the network from any given node. This is implemented as a single-tier hierarchy as well as a recursive multi-tier hierarchy.

5.1 Single-Tier Centralization

In a single-tier centralization approach, a single iteration of the centralization greedy algorithm assigns k sub-APs based on highest overall proximity where k is a function of the total number of hosts on the network. Figure 5.1 shows topology viability metrics for an implementation of this single-tier centralization greedy algorithm.

FIGURE 5.1 – TABLE OF STATS FOR IAN ALGORITHM

5.2 Multi-Tier Centralization

In a multi-tier centralization approach, the centralization greedy algorithm recurses up to $\log_2 k$, where k is a function of the total number of hosts on the network, and on layer n of recursion, the algorithm assigns $\frac{k}{2^n}$ subAPs, where $n = 0$ at the first layer of recursion. Results from the multi-tier centralization algorithm can be found in figure 5.2, and pseudocode for the algorithm is shown below.

FIGURE 5.2 – TABLE OF STATS FOR RECURSIVE IAN ALGORITHM

Algorithm 3 Recursive Centralization Greedy Algorithm

Require: *hosts* is a list of all hosts on the network, *subAPs* is an empty list of subAPs

procedure BUILDNETWORK(*hosts*, *subAPs*)

sort hosts by centralization

for $i \leftarrow 1$ to k **do**

$subAPs[i] \leftarrow host[i]$

for $i \leftarrow k$ to $numHosts$ **do**

sort *subAPs* by distance to $host[i]$

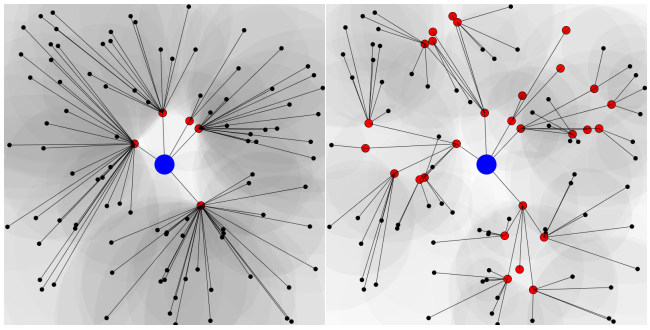
$host[i].myAP \leftarrow subAPs[0]$

$subAPs[0].myChildren.append(hosts[i])$

for *subAP* in *subAPs* **do**

buildNetwork(*subAP.myChildren*, [])

Figure N shows a comparison of the two algorithms for a randomly distributed data set. While the single-tier centralization reduces interference mostly around the physical AP, the recursive version optimizes overall interference.



(a) Un-Optimized (b) Optimized

Figure 6: Single vs Multi-tier centralization