On Topology Creation For An Indoor Wireless Grid

Gautam Bhanage, Yanyong Zhang and Ivan Seskar
Department of Electrical and Computer Engineering, Rutgers University
Wireless Information Network Lab (WINLAB), New Jersey
{gautamb, yyzhang, seskar}@winlab.rutgers.edu

ABSTRACT

Most space constrained indoor wireless testbeds have a majority of the nodes under a single collision domain. Simulation of multi-hop topologies on such testbeds is usually achieved either through the use of additional hardware like external noise injection or software techniques such as MAC frame filtering. We show that while these two techniques are able to create simple scenarios, creating large-scale, realistic topologies remains a challenge. In this paper, we propose and implement a Bursty PER filtering mechanism, which can filter packets on a link according to preset PER in the software domain. We first show that this new method can provide a better support for fine-grained link PER and localized link control. Further, we show that it is possible to create large scale topologies (of the order of 100 nodes). We substantiate our claims by empirical evaluation on the ORBIT indoor wireless testbed.

Categories and Subject Descriptors
C.2.1 [Computer Systems Organization]: Computer-Communication Networks, Network Architecture and Design; D.2.11 [Software Architectures]

General Terms
Experimentation, Design

Keywords
Wireless topology creation, software architecture, packet filtering

1. INTRODUCTION

Recently, testbed emulation is becoming an important evaluation platform [5, 21, 6] for the wireless networking community as simulation can provide more realistic details than simulations while avoiding the difficulty involved in maintaining and procuring access to real network deployment. ORBIT [5] is a laboratory-based wireless network emulator which uses a 20m × 20m two-dimensional grid of 400 802.11 radio nodes with reproducible wireless channel models. Each node on the wireless grid is a full computer capable of carrying any disk image, along with a wired control and experimental wired and wireless interfaces.

ORBIT has a single collision domain, which dictates the need for explicit topology mapping mechanisms to create multi-hop topologies. In this study, we discuss the insufficiencies of the standard topology creation tools i.e. noise generation and MAC frame filtering. Noise generation works by injecting noise at the receivers of certain nodes such that they are able to receive packets only from a controlled subset of nodes. Current noise generation approach provides a coarse control for topology creation. Not only is the mapping process significantly time consuming but it also limits the number of topologies that may be created with the current setup. On the other hand, MAC frame filtering provides considerably easy mapping of experiments with software control. However, topologies created with such an approach lack the realism seen with links in terms of packet drop characteristics because this scheme can show a radio link as either being on or off. Studies such as [4, 13] show that intermediate link PERs exist cannot be emulated with such an approach. In order to address these insufficiencies, we introduce a new method which is similar to MAC frame filtering, but it gives extensive control of the packet drop characteristics, allowing the experimenter to mimic bursty channel errors and time varying PER of links through software control. We have implemented and tested this through a Linux kernel module on the ORBIT grid. We show that
the new method can be used to create topologies that were not possible with the two standard techniques.

The rest of the paper is organized as follows. Section 2 provides a detailed discussion on existing topology generation tools. Section 3 describes the design and implementation of our topology creation tool. Section 4 compares our approach with other topology creation techniques. Section 5 applies our tool to create large-scale topologies and Section 6 provides the concluding remarks.

2. RELATED WORK

Wireless topologies can be emulated by wired testbeds or wireless testbeds. For example, the EMPOWER emulator [20] models a wireless mobile node on a wired testbed. The downside of emulating wireless links using wired networks is that it leaves out the details from the physical and the medium access layer such as the effects of propagation, interference, multi-path, physical layer capture, and overheads of the MAC itself.

At the same time mapping real world multi-hop topologies on space constrained wireless testbeds in an efficient and accurate manner presents a significant challenge. The following techniques have been proposed to create a multi-hop wireless topologies: (1) Transmit Power control, (2) Physical shielding/Attenuation, (3) Noise generation, and (4) MAC filtering. Next, we will discuss the insufficiencies of these techniques in detail.

2.1 Power Control/RF Attenuation

Transmit power control has been used by [14] to create wireless multi-hop topologies. The lowest transmit power with currently deployed off-the-shelf 802.11 WLAN cards (ATHEROS 5212 based chipset) is 1mW. Hence with an indoor path loss exponent of 2, the distance between any two nodes that cannot hear each other would be considerably large, making indoor deployment difficult. To make matters worse, COTS WLAN cards are only becoming better with sensitivity of up to $-92\text{dBm}$. These limitations with transmit power settings make the use of attenuators necessary.

Radio shielding or change of attenuation (if expensive programmable attenuators are not available) requires human intervention. Clancy et. al [21] show that it is possible to create multihop topologies with controlled attenuation through programmable RF matrix switches. The MiNT testbed [6] has a similar approach to topology creation with RF attenuation and power control. Both solutions share the drawback of either incurring high cost due to the use of RF matrix switches and programmable attenuation or severely limiting the number and type of emulated topologies with fixed attenuators without human intervention. The need of human intervention eliminates any chances of the testbed supporting remote experimentation. Therefore, we do not discuss this method further in the rest of this paper as it is not a practical solution in our current ORBIT design.

2.2 Noise Injection

Increasing the noise floor in certain parts of the network provides another way of topology emulation [15]. In this approach, noise is injected in various parts of the network, and an exhaustive search is needed to find the appropriate set of nodes whose resulting PER values are within the specified range. This search is not only time consuming but also it is not always possible. The limitation on the number and type of topologies that can be created through this approach has been discussed in [11]. Further, [15] shows it is possible to create simple string topologies with 4 to 6 hops, but creation of arbitrary topologies is still very difficult, especially when the number of noise sources are limited. Also, as noted by [21], signal reception is not always a function of the SNR but of absolute power. Consequently, approaches that lower the SNR by increasing the noise floor may not provide the desirable results.

Narrow band noise generation also has a channel dependency. For example, current noise generators used by the ORBIT testbed are capable of generating noise with a 40Mhz bandwidth only. Hence experiments that test multihop protocols that are across multiple frequencies and multiple channels such as those in [12], cannot be emulated. Finally, another problem with this approach was pointed out in [15] that the repeatability is not guaranteed when nodes with marginal SNR values are involved in experimentation. Marginal SNR values could result in the nodes having varying PER values in consecutive runs of the experiment.

2.3 MAC Frame Filtering

MAC frame filtering is a software based approach to topology creation, wherein a node drops MAC frames from the sources whose MAC addresses are on its filtering list. Such an approach promises to create arbitrary topologies. Standard open source tools and fire-walling utilities such as ebtalbes[7], iptables[10] and mackill[17] can be used for filtering MAC frames. The disadvantage for address filtering is that it cannot emulate many real world scenarios such as asymmetric links, variable PER links, and mobility, since it assumes a link is either completely up (PER of 0%) or down (PER of 100%). This feature is also shared by the Click modular router [16] which is capable of limiting the throughput but does not allow control of link PERs.

2.4 Topology Generation Tool Requirements

Below we summarize the requirements of a good topology generation tool:

1. Simple node mapping
2. Reproducibility
3. Least human intervention
4. Realistic link characteristics: Approach should allow creation of links with varying, non-binary PER.
5. Non-intrusiveness: The emulation tool should not incur a large overhead in the observed performance. This can be achieved by using independent control and data channels.

3. PER BASED PACKET FILTERING

Conventional MAC filtering relies on discarding all MAC frames seen from a particular sender MAC address. MAC filtering is popular since it does not require external hardware (noise injection) or man power (moving RF shields) nor incurs extra cost for experimentation. The downside of this approach lies in the fact that the emulated links are binary, either ON (PER of 100%) or OFF (PER of 0%). As a result, we call the conventional approach on-off MAC
filtering. In this study we will extend the on-off MAC filtering technique to allow for continuous PER values between 0 and 1, and we refer to the proposed technique as PER based packet filtering.

3.1 Overview

The key idea with PER based packet filtering is to emulate partial packet losses on links by dropping MAC frames in direct proportion to the desired link PER. The PER based packet filtering engine is implemented by modifying the kernel patch for Mackill[17], an open source utility for topology creation. An on-off Mackill drops all packets seen from a particular interface. In our implementation we modified the kernel patch to drop packets based on a specified rate. Each interface on a node can now independently administer packet filtering at different filtering rates for each MAC address. The modified kernel patch was implemented and tested on a linux machine running a debian distribution with the 2.6.12 kernel. To ensure efficient implementation, special care was taken while using critical path instructions. The kernel patch requires netfilter support to be enabled and compiles as a kernel module.

3.2 Packet Drop Pattern

Different nature of links may be emulated by changing the packet dropping patterns on the links. Many channel error models rely on i.i.d distribution of channel errors. However, further research revealed that channel errors are bursty in nature due to inherent attenuation, inter-symbol interference (ISI), doppler shift and multipath fading [1]. Since initial efforts by Gilbert[9] and Elliot[8], bursty channel errors have been modeled by a simplified two state Markov model [22], where one of the states indicates an error and the other indicates a successful transmission. Since emulation of PER using a markov model in the kernel could incur substantial packet processing overhead we propose the use of mean error lengths and error free periods for controlling channel error burstiness. In fact, a trace based evaluation of channel error behavior in [19] evaluated the mean and standard deviation of packet error probability, error length, and error free lengths which could be useful in topology creation with our approach.

Using our filtering scheme, we need to specify the link PER which is a fraction of the total number of dropped and passed packets as:

\[ \frac{N_{\text{dropped}}}{N_{\text{dropped}} + N_{\text{passed}}} \]

where \(N_{\text{dropped}}\) is the number of packets that will dropped in a sequence, and \(N_{\text{passed}}\) is the number of packets that are passed. For example, with \(N_{\text{dropped}} = 10\) and \(N_{\text{passed}} = 30\), the node will drop 10 packets in a row, and then pass the next 30 packets. While maintaining the same PER, the user can control the size of the drop burst by varying the values of \(N_{\text{dropped}}\) and \(N_{\text{passed}}\). For example, a PER of 10% can be achieved by dropping 1 packet out of every 10 packets or by dropping 10 packets out of every 100, but the implications of these two options are completely different when ad hoc routing is used because they indicate different

\[ A \rightarrow B \rightarrow C \rightarrow D, \text{ where } A, B, C, D \]

\[ \vdots \]

### Table: Experimental Parameters Used With ORBIT Nodes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Rate</td>
<td>6Mbps</td>
</tr>
<tr>
<td>Offered Load</td>
<td>Time Varying</td>
</tr>
<tr>
<td>Experiment Duration</td>
<td>2 Minutes</td>
</tr>
<tr>
<td>Averaging Duration</td>
<td>Per Second</td>
</tr>
<tr>
<td>Operation Mode</td>
<td>802.11a</td>
</tr>
<tr>
<td>Chipset</td>
<td>Atheros</td>
</tr>
<tr>
<td>Driver</td>
<td>MadWiFi 0.9.3</td>
</tr>
</tbody>
</table>

3.3 Applicability of PER Based Packet Filtering

Like any software topology mapping approach, PER based packet filtering focuses on creating large-scale complex network topologies, and some low level details may be lost. Specifically, we would like to emphasize that this scheme is more suitable when wireless protocols above MAC layers are studied.

First let us look at the problems with the PHY layer. Consider a network consisting of four nodes: \(A, B, C,\) and \(D\) with connectivity as: \(A \rightarrow B \rightarrow C \rightarrow D\), where \(A\) and \(D\) cannot hear from each other. However, mapping this topology on a testbed may result in all four nodes falling in a single collision domain. It is noted that even though MAC filtering enforces the topology by dropping frames, the nodes are still within each other’s interference range, leading to a decreased aggregate throughput and a noticeable increase in delay. MAC filtering techniques will also lose some unicast MAC-layer interactions. Since ACKs and re-transmissions for unicast transmissions happen in the hardware, it is impossible to model the effect of these in software.

4. BASELINE COMPARISON

The advantage of the PER based packet filtering technique over traditional on-off MAC filtering is obvious as the former facilitates more accurate emulation of link quality. In this section, we then focus on the comparison of the PER based packet filtering approach against the noise injection technique. Specifically, we look at important characteristics for a topology generation tool such as fine-grained PER control and localized control. Finally, we look at the emulation of asymmetric links as a case study.

4.1 ORBIT Wireless Testbed

ORBIT is a two-tier laboratory emulator/field trial network testbed designed to achieve reproducibility of experimentation, while also supporting evaluation of protocols and applications in real-world settings [5]. The setup consists of a large two-dimensional grid of 400 802.11 radio nodes which can be dynamically interconnected into specified topologies with reproducible wireless channel models. Every node is a small form factor PC with 1GHz Via C3 CPU, 512 MB RAM, 20 GB hard disk and three ethernet ports, one of which is used for node configuration and control. Major-
4.2 Fine-Grained PER Control

Since real world networks may witness a wide spectrum of PER values, it is important to be able to conveniently configure the emulations to achieve any specified PER values. To demonstrate whether the fine-grained PER control can be easily achieved by these approaches, we conducted experiments on a single wireless link between two nodes placed at arbitrary positions but within line of sight as shown in Figure 3(a). The nodes are configured in ad hoc mode and have static IPs. Other experiment parameters are shown in Figure 2. We vary packet size from 64 bytes to 1470 bytes and test the PER granularity of both approaches with an offered load of 2Mbits/sec.

Figure 4 plots the observed PER as a function of the injected noise level for different packet sizes. Noise levels were varied from $-44$ dBm to $-32$ dBm. These results show that the PER granularity for noise injection is not fine enough: the link PER increases from 0% to 100% within only 5 dBm of injected noise. In particular, in some cases, we observe a 60% PER difference when the injected noise level only goes up for 1 dBm. Further, we observe that PER for smaller packets increases faster than PER for larger packets. This phenomenon can be attributed to the fact that as the number of errors increases, there are more retransmissions for smaller packets. We note that this trend with packet size will be opposite if the same experiment was performed with an unsaturated channel. This further proves that PER mapping with noise injection is channel dependent, making it harder to control.

Figure 5 shows the observed link PERs by PER based packet filtering for different desirable PER values. Irrespective of the packet size, it can configure PER values by setting the appropriate ratio of $N_{\text{dropped}} / N_{\text{passed}}$. This comparison demonstrates that the PER based packet filtering technique can provide fine-grained PER control.

4.3 Localized Control

A topology mapping scheme should be able to change links locally without having an effect on the global topology. To
The reciprocity principle is based on the property that electromagnetic waves traveling in both directions will undergo the same physical perturbations such as path loss, reflection, refraction, and diffraction. Channels which do not have the reciprocity property are known as asymmetric channels. Previous studies [13] have shown that asymmetric wireless links are common in wireless networks. Asymmetric link emulation is important when evaluating efficiency of ad hoc routing protocols, optimal transmission power selection, and other higher-layer protocols. In this paper, symmetry of a channel is measured by the difference in PER between the forward and reverse link of a pair of nodes.

Figure 6 illustrates the experimental settings for both approaches. The experiment parameters are shown in Figure 2. In evaluating the noise injection technique, we chose the first node \((A)\) away from the noise source, and the other node \((B)\) near the noise antenna. Figure 7 shows the PER differences between the forward \((A \rightarrow B)\) and the reverse \((B \rightarrow A)\) links when the noise level increased from \(-44dBm\) to \(-39dBm\). Below \(-44dBm\) we did not observe any deterioration in link qualities. As the noise gradually increased, the forward link degraded, resulting in an increased PER difference. After the noise power increased beyond a particular level, \(-42dBm\) in our case we observe that the forward link completely broke and that the reverse link also significantly degraded. As a result, the PER difference between two links decreased. This result shows that it is hard to create asymmetric links by injecting noise as it is nearly impossible to contain noise’s effect locally without having more global effect.

To perform asymmetric link tests with the PER based packet filtering scheme, we used the setup as shown in Figure 6 with parameters as shown in Figure 2. We varied the desired PER values at both nodes and the measured PER difference between the two links is as shown in Figure 8. The result shows that this method can achieve any specified PER difference and can thus conveniently set up asymmetric links.

### 4.4 Summary Of Comparisons

A comprehensive comparison of the topology creation techniques is as summarized in Figure 10. It can be observed that the proposed PER based packet filtering provides considerable advantages over the other approaches with the strongest advantage being easy mapping of arbitrary link error rates for randomly selected nodes.

### 5. TOPOLOGY CREATION TESTS

Real world environments like those in a military scenario or with first responders require performance evaluation of ad hoc networks with a large number of nodes before they can be deployed. Comparative performance of wireless protocols with a medium scale networks of up to 23 nodes is shown in [2]. Using our scheme we will demonstrate its utility by first recreating a 6-node wireless system measured in [13], and then by generating topologies with up to 100 nodes. We begin with a brief introduction to our software framework.
<table>
<thead>
<tr>
<th>Ability</th>
<th>Noise</th>
<th>Shielding/Power Control</th>
<th>MAC – Filtering</th>
<th>PER – Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Dependence</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>PER Granularity</td>
<td>coarse</td>
<td>coarse</td>
<td>coarse</td>
<td>fine</td>
</tr>
<tr>
<td>Local Control</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Remote Execution</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Node Mapping</td>
<td>specific</td>
<td>any</td>
<td>any</td>
<td>any</td>
</tr>
<tr>
<td>Deployment Cost</td>
<td>significant</td>
<td>significant</td>
<td>free</td>
<td>free</td>
</tr>
<tr>
<td>Emulating RSSI</td>
<td>precise</td>
<td>coarse</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Figure 10: Summary of comparison of approaches for topology creation. Channel dependence reflects if the approach is limited to creating topologies on a specific channel only. Granularity determines fineness of link PER control. Local control determines if local changes in topology creation affect other parts of the network. Remote execution determines if it is possible to automate and execute the scheme remotely. Node mapping determines if random nodes can be used for topology mapping. Deployment cost is indicative of relative costs of deploying each of the schemes. Mapping RSSI indicates whether the scheme can emulate realistic link SNRs.

An automatic, reproducible experimental setup requires that most tasks be executed through automation. Typically creation of topologies with hundreds of nodes requires an efficient architecture for setup and execution of experiments. In this section we provide a brief overview of our setup on the ORBIT testbed.

The basic experiment setup is as shown in the figure 9. Our approach relies on most of the features provided by the ORBIT experimental setup. The experimenter is able to remotely control the experiments by logging into the gateway machine in the ORBIT setup. Internally the grid is controlled by server machines running control and monitoring applications. On each and every node on the grid, control commands issued by the application servers are executed through the ORBIT control framework.

The core software for topology generation is built using C++ and perl. This software is also capable of generating arbitrary topologies with varying number of nodes, node-connectivity, and number of flows. The core part of the software is the kernel module implemented in C that drops packets as discussed in Section 3.

The automated topology creation process is best explained by figure 12. The user can specify experiment parameters such as number of nodes, flows, node connectivity and the nature of links (random or as per a fixed topology) through an experiment control and layout script. The software parses this script and determines the requirements such as the required number of nodes and types of cards required for successfully executing the experiment. This requirement is verified with an inventory service which maintains a list of the number, type of machines on the ORBIT grid and other useful information such as MAC addresses of cards which will be useful in setting up filtering. Based on this information, the software performs node mapping and generates control scripts for configuring radios on the nodes. These scripts are responsible for loading the wireless driver, configuring radio interfaces, assigning IPs, setting up PER based packet filtering, and finally starting the routing and generating traffic. All scripts are automatically delivered and executed on the selected nodes using `pssh`[3]. Finally, logs are collected, parsed, and made available to the user. This systematic approach allows us to store and reproduce all of our topologies.

![Table showing comparison of approaches for topology creation](image)

<table>
<thead>
<tr>
<th>Node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.188</td>
<td>1.51</td>
<td>0.24</td>
<td>0.48</td>
<td>-0.09</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-9.17</td>
<td>0</td>
<td>-30.55</td>
<td>0.45</td>
<td>-18.29</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.56</td>
<td>5.28</td>
<td>0</td>
<td>-16.11</td>
<td>3.55</td>
<td>1.76</td>
</tr>
<tr>
<td>4</td>
<td>1.18</td>
<td>-3.07</td>
<td>-3.38</td>
<td>0</td>
<td>3.92</td>
<td>5.35</td>
</tr>
<tr>
<td>5</td>
<td>0.44</td>
<td>-0.58</td>
<td>-0.24</td>
<td>0.78</td>
<td>0</td>
<td>4.49</td>
</tr>
<tr>
<td>6</td>
<td>0.63</td>
<td>0.78</td>
<td>0.07</td>
<td>0.18</td>
<td>4.58</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 11: Normalized % error in observed link throughput. Link PERs are set according to the percentage of successful probe packets based on indoor measurements.

### 5.1 Software Architecture

An automatic, reproducible experimental setup requires that most tasks be executed through automation. Typically creation of topologies with hundreds of nodes requires an efficient architecture for setup and execution of experiments. In this section we provide a brief overview of our setup on the ORBIT testbed.

The basic experiment setup is as shown in the figure 9. Our approach relies on most of the features provided by the ORBIT experimental setup. The experimenter is able to remotely control the experiments by logging into the gateway machine in the ORBIT setup. Internally the grid is controlled by server machines running control and monitoring applications. On each and every node on the grid, control commands issued by the application servers are executed through the ORBIT control framework.

The core software for topology generation is built using C++ and perl. This software is also capable of generating arbitrary topologies with varying number of nodes, node-connectivity, and number of flows. The core part of the software is the kernel module implemented in C that drops packets as discussed in Section 3.

The automated topology creation process is best explained by figure 12. The user can specify experiment parameters such as number of nodes, flows, node connectivity and the nature of links (random or as per a fixed topology) through an experiment control and layout script. The software parses this script and determines the requirements such as the required number of nodes and types of cards required for successfully executing the experiment. This requirement is verified with an inventory service which maintains a list of the number, type of machines on the ORBIT grid and other useful information such as MAC addresses of cards which will be useful in setting up filtering. Based on this information, the software performs node mapping and generates control scripts for configuring radios on the nodes. These scripts are responsible for loading the wireless driver, configuring radio interfaces, assigning IPs, setting up PER based packet filtering, and finally starting the routing and generating traffic. All scripts are automatically delivered and executed on the selected nodes using `pssh`[3]. Finally, logs are collected, parsed, and made available to the user. This systematic approach allows us to store and reproduce all of our topologies.

### 5.2 Small Scale Network

To highlight the effectiveness of our approach we first tried to reproduce the indoor environment measured in [13], in which a 6 node network was measured across 3 homes in US and UK. Link SNR values in our emulated topology were mapped to the ratio of successful probes seen in the original measurements. We measured throughput on our emulated topology and calculated the normalized percentage difference with respect to their throughput measurements. Figure 11 shows the normalized difference in measurements for every pair of nodes. We found that 87% of the links were accurately emulated with normalized differences below 5%. The few large differences are because the assumed correlation between the ratio of successful probes and the observed throughput in the original study does not always hold. We note that a setting like this could be valuable to evaluate routing protocols. An example scenario would be one in which we run routing protocols over the existing setups and compare arbitrary flow throughput values across the three homes. Figure 13 shows the results of a sample test with two flows whose throughput values are compared across the three homes with AODV and OLSR routing protocols. The throughput difference is related to route selection and hence performance of each of the routing algorithms. In this case, the throughput numbers are not of prime importance, but rather the ability to recreate the settings and measure performance with arbitrary protocols makes our approach...
5.3 Large Scale Network

Comparing the performance of routing protocols across multiple random topologies ensures that we do not favor any one topology for testing. For example earlier in cases where links are asymmetric OLSR may perform better than AODV, but this may not be true in other situations. To test the ability to generate random topologies, we created four random topologies: (1) 5 nodes, 1 flow, (2) 20 nodes, 2 flows, (3) 50 nodes, 5 flows, and (4) 100 nodes, 10 flows. On average, every node in each of the networks is connected to 30% of the other nodes in the network. The maximum flow length was limited to 3. Aggregate offered load for each of the topologies was distributed evenly across all flows and maintained at 1Mbits/sec. AODV[18] was used and other experiment parameters are same as shown in Figure 2.

Figure 14 shows the results obtained for the different topologies with different link PER values. Using our framework we could create large arbitrary topologies as long as sufficient hardware was available. Some example studies we can have on these topologies are listed below:

1. We can emulate topologies with different distances between nodes. For example, we could increase the PER values on existing links to emulate the nodes being further apart in space, a feature which we call artificial stretching. Figure 14 shows the results with 0%, 30% and 50% link PER with the same topologies.

2. We can study detailed performance of routing protocols such as how well it can survive the loss of control packets.

Though the goal of our study is not to quantify the performance of AODV[18], but to demonstrate the utility of the PER based packet filtering technique, we present some observations that could be worth of further investigation:

- For random topologies with up to 35 nodes with none or marginal packet loss the AODV routing algorithm fares well and all the flows complete successfully.

- As the number of nodes increases (say, to 50 or 100) we observe an increase in the routing overhead of up to 22%.

- AODV does not prioritize control packets and hence loss of routing information results in breakage of end to end connectivity with only flows across adjacent nodes completing. The bursty packet dropping pattern aggravates the problem even more when consecutive routing packets are dropped.

6. CONCLUSION

A detailed qualitative and quantitative comparison of existing topology generation tools shows that there are limited ways to achieve fine granularity, easy control, remote execution support, and repeatability. We propose a PER based filtering scheme to address these needs when recreating topologies on an emulation testbed from real life. A thorough comparison of the methods provides an insight into the value of our approach. We validated our approach by generating topologies from earlier measurements, and tested sample topologies with up to 100 nodes.

<table>
<thead>
<tr>
<th>Setting</th>
<th>AODV[kbps]</th>
<th>OLSR[kbps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home1</td>
<td>154 + 73.1</td>
<td>151 + 133</td>
</tr>
<tr>
<td>Home2</td>
<td>423 + 0</td>
<td>277 + 0</td>
</tr>
<tr>
<td>Home3</td>
<td>67.6 + 228</td>
<td>50 + 278</td>
</tr>
</tbody>
</table>

Figure 14: Throughput (kbps) results for flows in topologies of varying size with varying packet drop rates. Routing is done with AODV.
7. REFERENCES


