Generating Network Topologies That Obey Power Laws

Christopher R. Palmer
`crpalmer@cs`

J. Gregory Steffan
`steffan@cs`

15-744 Networks Project

Abstract

Recent studies have shown that Internet graphs and other network systems follow power-laws. Do artificial network topologies commonly used in network simulation studies obey these power-laws? In this paper we show that current topology generators do not obey all of the power-laws, and we present two new topology generators that do. Finally, we re-evaluate a multicast study to demonstrate the impact of power-law topologies.

1 Introduction

Recent studies have shown that Internet graphs follow power laws [5]—i.e., certain graph metrics follow the distribution

$$y \propto x^{\alpha}$$

This phenomenon has been observed in router topology, inter-domain topology [5], and the world-wide-web [1, 2, 6, 7, 8]. Previous metrics used to characterize Internet graphs have focussed on averages: for example, average out-degree of routers. While these metrics are important, they do not capture higher-order properties such as the power-laws.

Network studies often simulate artificial network topologies to evaluate new ideas and schemes. There are several topology generation algorithms that are commonly used, but these are parameterized by average-based metrics and do not necessarily produce topologies that follow the observed power laws. It is therefore possible that the artificial networks used in simulation-based studies are not "realistic", and that the corresponding results are inaccurate.

We address this issue by answering the following three questions. First, *do current topology generators used in network simulations produce artificial networks that obey power laws*? We show that commonly used topology generators do not. Second, *how do we generate graphs that obey power laws*? We present two algorithms and compare their power-law performance with that of the current topology generators. Third, *what impact do power-law topologies have on simulation studies*? We re-evaluate the STORM [10] multicast scheme with both the current and new topology generators, and show that topology does impact results.

The remainder of this paper is organized as follows. In the next section, we review the power-laws and evaluate current topology generators against them. In Section 3, we present and evaluate two algorithms for generating power-law topologies, and in Section 4 we explore the impact of power-law topologies on the STORM multicast scheme. Finally, we conclude in Section 5.

2 Power Laws and Current Topology Generators

In this Section, we describe in detail the power-laws defined by Faloutsos *et al* [5]. Then, we explain the commonly used topology generators. Finally, we show that these topology generators do not follow all of the power-laws.

2.1 The Faloutsae Power-Laws

Faloutsos *et al* [5] define four power-laws: rank exponent, out-degree exponent, hop-plot exponent, and eigen exponent. [1] For convenience, we repeat the definitions here.

Power-Law 1: rank exponent The out-degree, d_v, of a node v, is proportional to the rank of the node, r_v, to the power of a constant, $\mathcal{R}$:

$$d_v \propto r_v^{\mathcal{R}}$$

This power-law is evaluated by computing the out-degree for every node, sorting the out-degrees in descending order, and plotting in log-log space. Data for inter-domain graphs and router graphs show a near-linear distribution, with a slight deviation at either end.

Power-Law 2: out-degree exponent The frequency, f_d, of an out-degree, d, is proportional to the out-degree to the power of a constant, $\mathcal{O}$:

$$f_d \propto d^{\mathcal{O}}$$

Power-Law 3: hop-plot exponent The total number of pairs of nodes, $P(h)$, within h hops, is proportional to the number of hops to the power of a constant, $\mathcal{H}$:

$$P(h) \propto h^{\mathcal{H}}, h \ll \delta, \text{ the diameter}$$

Power-Law 4: eigen exponent The eigenvalues, λ_i, of a graph are proportional to the order, i, to the power of a constant, $\mathcal{E}$:

$$\lambda_i \propto i^{\mathcal{E}}$$

Faloutsos *et al* suggest using these power-laws to measure the realism of synthetic graphs [5]. To test these measures we use linear correlation and visual inspection. This will be discussed in more detail in Section 2.3.

[1] This is not actually deemed a power-law by Faloutsae, but we will call it so for simplicity.

2.2 Current Topology Generators

Topology generators currently used in simulation studies are constructed to obtain a desired average out-degree, and may even produce an exact hierarchy of sub-graphs. While these techniques make intuitive sense, they do not necessarily capture the power-law properties measured in real internet graphs. For convenience, we now define the two most common topology generation schemes.

Waxman Waxman's [9] graph generation algorithm places random points in cartesian two-space. An edge is added between two points u and v with a probability inversely proportional to the distance between them, as given by

$$P(u, v) = \alpha e^{-d_{u,v}/\beta L}$$

where α and β are parameters, $d_{u,v}$ is the distance between u and v, and L is the maximum inter-node distance.

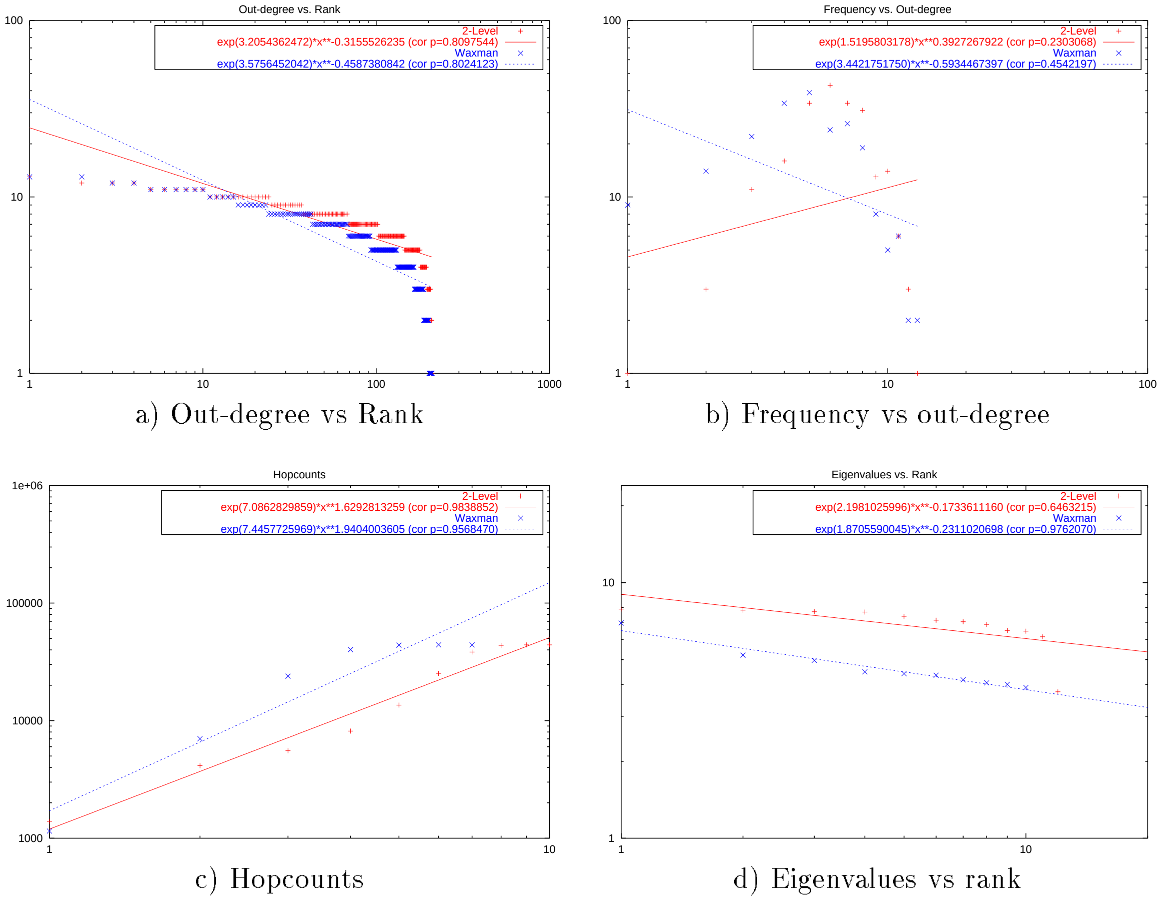
N-level Hierarchical N-level hierarchical [4, 11] methods generate a collection of random graphs and connect them in a hierarchy. There are many variants of this approach, including some that add extra random connections across hierarchy levels.

2.3 Performance on Power-Laws

Power-laws define a line in log-log space. We evaluate the performance of the existing topology generators with respect to the power-law metrics by measuring the strength of a linear relationship in log-log space. Specifically, a scatter plot is constructed from the data, a line of minimum least square errors is plotted and the linear correlation of the log transformed data is measured. We report absolute values of the correlation, ρ. This is equivalent to measuring the quality of the least squares line.

Random networks are generated using Waxman's algorithm and the 2-level algorithm used in the STORM [10] study. Network generation parameters are selected that correspond to their *200 client* simulation (approximately 200 nodes with an average out-degree of approximately 4). The power-law graphs for the four metrics are shown in Figure 1.

The hopcounts and the eigen values graphs (Figures 1c and 1d respectively) show strong power-law relationships. The correlations are mostly high ($\rho \geq .95$) except for the 2-level eigen values metric ($\rho = .65$). Visual inspection reveals a strong power-law relationship with an outlyer affecting the correlation. The degree based graphs (Figures 1a and 1b) show no power law relationships. In particular, the out-degree vs. rank graph ($\rho = .8$) is definitely concave and not linear. The frequency vs. out-degree graph ($\rho < .5$) appears to define a Gaussian relationship in log-log space. To summarize, neither Waxman's algorithm nor the 2-level hierarchical algorithm generate graphs that pass all of the power-law tests.



a) Out-degree vs Rank

b) Frequency vs out-degree

c) Hopcounts

d) Eigenvalues vs rank

Figure 1: Power-law plots for current topology generators.

3 New Algorithms for Generating Power-Law Topologies

In this section, we present two new algorithms for generating network topologies that do obey power-laws. The first method uses a power-law to guide graph construction, while the second method uses a recursive probability distribution. We evaluate these new methods on the four power-laws.

3.1 Network Graphs

A network is a set of hosts with network links between some subset of them. We can model such a structure as an undirected graph $G = (V, E)$ where $E \subseteq V \times V$ such that $(u, v) \in E \Rightarrow (v, u) \in E$. We further require that $(u, u) \notin E$ for all u (i.e., there are no hosts that have an external link to themselves). A common representation of an undirected graph is the N by N adjacency matrix $A_{u,v}$.

```
PLOD(N, M, A) =
    Let N be the number of nodes
    Let M be the number of edges
    Let A be the adjacency matrix

    FOR i : 1..N
        x = uniform_random(1, N)
        out_degree_i = βx^{-α}

    FOR i : 1..M
        WHILE 1
            r = uniform_random(1, N)
            c = uniform_random(1, N)
            IF r ≠ c AND out_degree_r AND out_degree_c AND !A_{r,c}
                out_degree_r − −, out_degree_c − −
                A_{r,c} = 1, A_{c,r} = 1
                BREAK
```

Figure 2: Power-law out-degree generator

3.2 Power-Law Out-Degree Algorithm

A simple way to generate a graph that obeys a power-law is to use that power-law to guide the construction of the graph. Node out-degree is a straight-forward property to manipulate (as opposed to hop-count), so our first algorithm constructs a graph from a collection of nodes that have a power-law distribution of out-degrees. We call this the *Power-Law Out-Degree* algorithm, or PLOD.

The algorithm for PLOD is shown in Figure 2. First, we obtain a number of *out-degree points* for each node in the graph by sampling a random exponential distribution. Second, we place edges in the adjacency matrix for the graph such that every node obtains the assigned out-degree. The edge placement loop picks a random pair of nodes and assigns an edge if each node has remaining out-degree points and there is not already an edge between the nodes. When an edge is placed between a pair of nodes, we also decrement the out-degree points for each node in the pair. If this initial pair of nodes is not suitable for an edge, we continue picking random pairs of nodes until we find one that is suitable [2].

The selection of parameters α and β for PLOD is important, as they will produce vastly different distributions of node out-degrees. We also want to obtain a certain number of edges in the generated graph. The value of β controls the y-intercept of the log-log plot ($x = 1$, the largest possible out-degree). Once we have chosen β, we select a value of α that produces the

[2] For performance reasons, our implementation simply picks pairs of nodes in order from the initial pair of nodes until an edge is placed

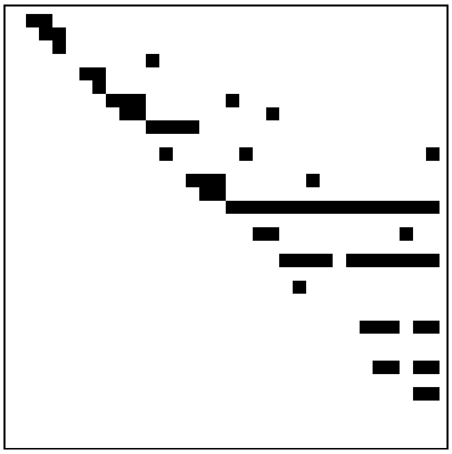


Figure 3: Example adjacency matrix generated by PLOD (shows upper triangle only)

desired number of edges [3].

3.3 Example

Figure 3 shows the upper triangle of the adjacency matrix for a 32 node graph generated by PLOD with the parameters α=0.685 and β=20.0. It is evident that some nodes have high out-degree, while most have low out-degree. Figure 4 shows the resulting network graph with 32 nodes and 71 edges.

We evaluate performance of PLOD on the Faloutsae power-laws at the end of this section, but first we introduce our second algorithm for generating power-law topologies.

3.4 Recursive Topology Generator

PLOD produces a binary adjacency matrix indicating the presence or absence of a link. The Recursive generator produces an adjcency matrix with real valued entries. Each real value (edge weight) can be used to represent some encoding of network capabilities. For example, edges with high weight could be treated as high bandwidth links in a simulation. There are two parts to Recursive. First, we define a probability distribution function that will randomly select a pair of nodes. Second, we use this probability density function to produce a network graph with real valued edge weights.

The probability density function is a generalization of an 80-20 distribution. An 80-20 distribution splits the space into two halves and places 80% of the values in the left half and 20% of the values in the right half. For a 2-dimensional array we defined the quantities

$$\begin{bmatrix} \alpha & \beta \\ \gamma & \epsilon \end{bmatrix}$$

($\alpha + \beta + \gamma + \epsilon = 1$) to correspond to the percentage of values we want to place in each recursive sub-array. We must take care to generate a symmetric matrix and we must ensure

[3]For this paper we have implemented an iterative search for α, although a closed-form solution likely exists

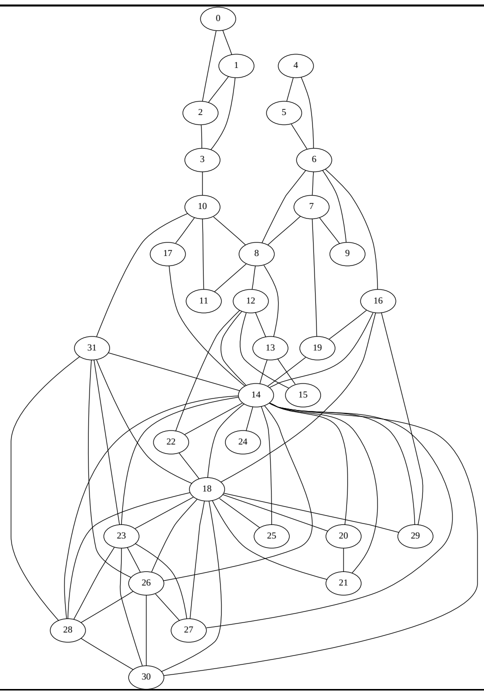


Figure 4: Example topology generated by PLOD

that the diagonal contains no links (only 0s). Careful accounting of the number of non-zero entries is required. For example, consider the 4 by 4 symmetric matrix. There are 12 potentially non-zero values, but only 2 of them are in the upper left quadrant:

$$\left[\begin{array}{cc|cc} 0 & - & - & - \\ & 0 & - & - \\ \hline & & 0 & - \\ & & & 0 \end{array}\right]$$

Also, notice that putting a link in the upper right quadrant is equivalent to putting a link in the lower left quadrant (due to symmetry). Now, if we wanted an $\alpha\%$ chance of putting a link in the upper left quadrant, then we must find a probability, p_1, such that the expected number of links in that quadrant is α. That is,

$$p_1 \cdot N_1 = \alpha \Leftrightarrow p_1 = \frac{\alpha}{N_1}$$

given that there are N_1 potential links in the quadrant. Repeating this argument (assuming N_2 potential links in the upper right/lower left quadrants) we need to find probabilities

$$\begin{aligned} p_1 &= \kappa \tfrac{\alpha}{N_1} \\ p_2 &= \kappa \tfrac{\beta+\gamma}{N_2} \\ p_3 &= \kappa \tfrac{\epsilon}{N_1} \end{aligned}$$

where κ is a normalization factor (probabilities sum to 1). It is relatively easy to verify that

```
gensym(n) =
    IF n = 2 THEN return (1, 2)

    Let N_1 = (n/2)^2 − (n/2) be # non-zero in upper left quadrant
    Let N_2 = (n/2)^2 be # non-zero in upper right quadrant
    Let κ = 1/((1/N_1)·α + (1/N_2)(β + γ) + (1/N_1)ε) be the normalization factor
    Let A = [ A_11  A_12 ]
            [ A_21  A_22 ]

    THEN WITH PROBABILITY
        α · κ/N_1 ⇒ RETURN gensym(n/2)
        (β + γ) · κ/N_2 ⇒ RETURN gen(n/2) + (0, n/)
        ε · κ/N_1 ⇒ RETURN gen(n/2) + (n/2, n/2)

gen(n) =
    IF n = 1 THEN return (1, 1)
    WITH PROBABILITY
        α ⇒ RETURN gen(n/2)
        β ⇒ RETURN gen(n/2) + (0, n/2)
        γ ⇒ RETURN gen(n/2) + (n/2, 0)
        ε ⇒ RETURN gen(n/2) + (n/2, n/2)
```

Figure 5: Recursive edge generator

$$\kappa = \frac{1}{\frac{1}{N_1}(\alpha + \epsilon) + \frac{1}{N_2}(\beta + \gamma)}$$

Figure 5 provides an algorithm, *gensym*, that will randomly select an edge, (u, v), from an n by n symmetric matrix using the probability distribution function we have defined. The selected edge will always be in the upper triangle. The probability density function defined by this code is shown in Figure 6. The darkness of a cell is an indicator of its relative probability.

Our topology generator takes as input parameters N and M (the number of nodes and edges, respectively) and generates a weighted graph using the edge generator. The graph is generated by repeatedly generating a link and incrementing the weight of that link until such time as we have M links. The details are given in figure 7. An example using the probability density function of figure 6 that generates 10 edges is show in 8. Again, darker entries are weighted more heavily than lighter entries.

It is up to the user to decide how to map the weightings to network capabilities. For example, the experimental results presented in the next section have two classes of links: high latency and low latency. We have defined high latency links to be the more heavily weighted

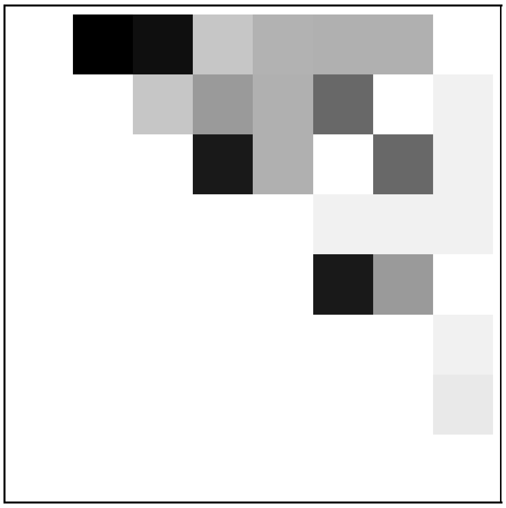


Figure 6: Probability density function for a 16 x 16 adjacency matrix

---

Recursive(N,M) =
    Let A be an N by N zero matrix
    WHILE A has fewer than M non-zero entries DO
        $(u, v) = gensym(N)$
        $A_{uv} = A_{vu} = A_{uv} + 1$
    DONE
    RETURN A

Figure 7: Recursive Topolgy Generator

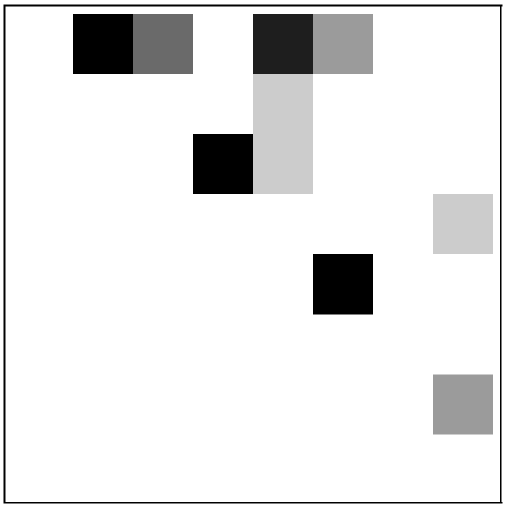


Figure 8: Weighted links generated for 16 node, 10 edge topology

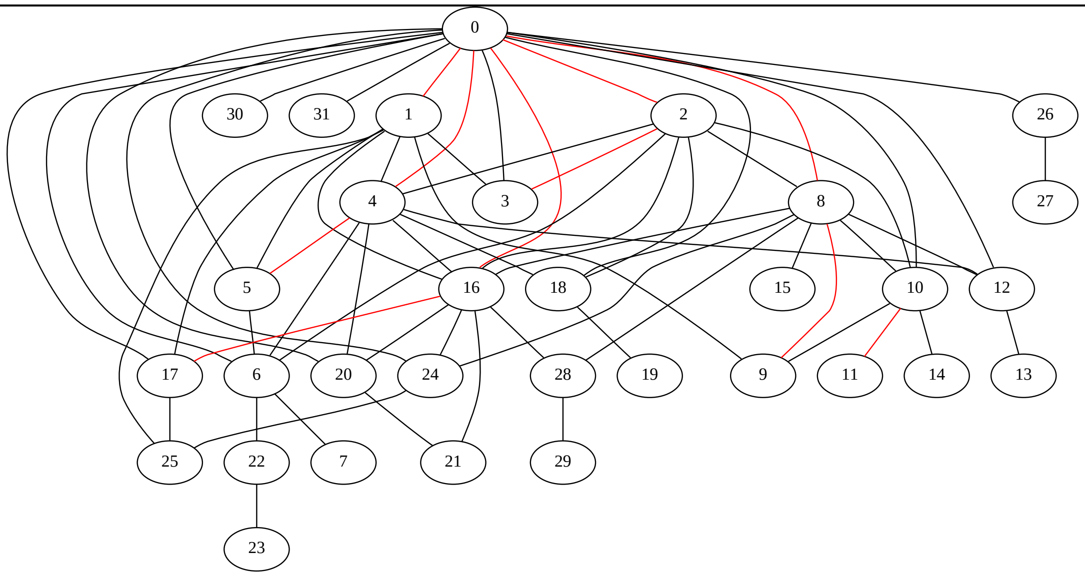


Figure 9: Example with 32 nodes, 50 low latency links (black) and 10 high latency links (red)

ones (assuming that weight corresponds to some measure of the traffic passing through a link). Using this idea, we generated a 32 node graph with 50 low latency links and 10 high latency links. This graph is shown in figure 9 and the high latency links are drawn in red.

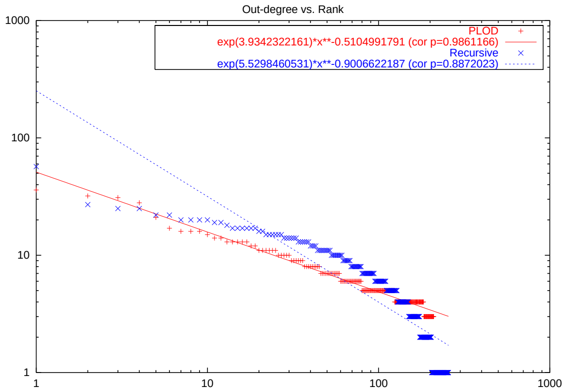
3.5 Evaluation

PLOD and Recursive are evaluated using the parameters and methodology presented in Section 2.3—the four power-law graphs appear in figure 10. PLOD has a nearly perfect out-degree vs. rank plot while Recursive appears as a characteristic 80-20 curve (figure 10a)). The 80-20 curve is similar to the results discovered by Bauer [3] using data collected by the Internet Mapping Project. All other metrics (figures 10b)-d)) exhibit excellent power-law relationships ($\rho \geq .92$).

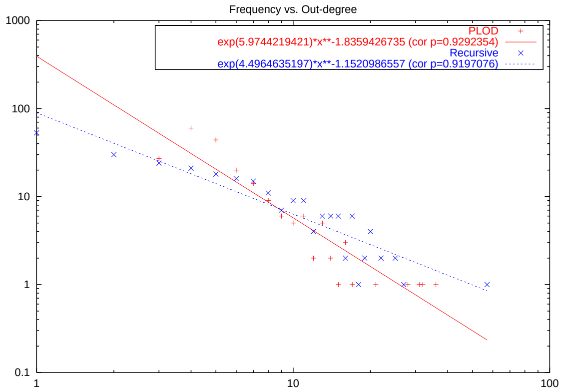
PLOD passes the Faloutsos power-law tests and Recursive passes all tests with a slight variation in the out-degree vs. rank plot. For comparison, table 1 summarizes the correlations that were discovered for the four generation algorithms. We conclude that PLOD and Recursive capture Internet characteristics far better than either Waxman or 2-level.

4 Impact of Power-Law Topologies

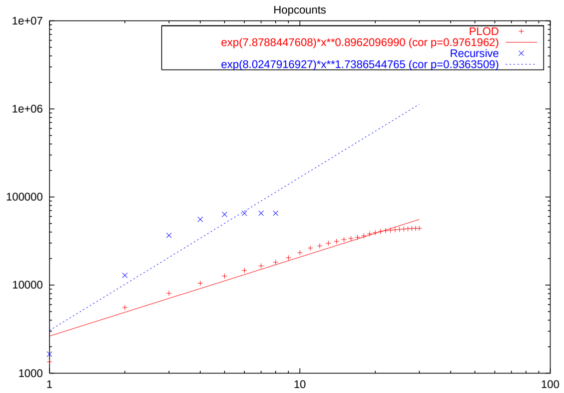
In this section, we investigate the impact of power-law topologies on simulation results. In particular, we re-evaluate the STORM [10] multicast scheme and show that results do vary by topology. We give a brief summary of the STORM multicast scheme, describe how we ensure that network graphs produced by the different topology generators are comparable, and finally we present the results of our re-evaluation of STORM.



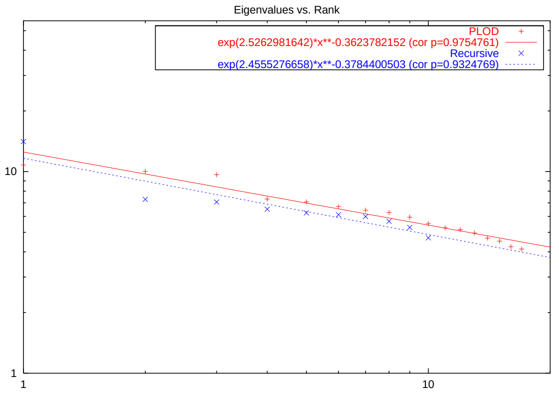
a) Out-degree vs Rank



b) Frequency vs out-degree



c) Hopcounts



d) Eigenvalues vs rank

Figure 10: Power-law plots for our new topology generators.

Table 1: Power-Law Summary: Correlations

	PL #1: Degree	PL #2: Deg. Freq.	PL #3: Hops	PL #4: Eigenval
2-Level	.81	.23	.98	.65
Waxman	.80	.45	.96	.97
PLOD	.99	.93	.98	.98
Recursive	.89	.92	.94	.93

4.1 The STORM Multicast Scheme

The STORM multicast algorithm [10] provides a scalable method of recovering from lost packets of a multicast stream. Each receiver (*client*) chooses a *parent* node from a set of neighboring multicast participants. When a packet is late, the client sends it's parent a *nack* packet. The parent then responds with a *repair* packet which, if received in time, is used to replace the original missing packet.

Simulation was used in the STORM study to show that the protocol overheads scale as the number of clients increased, and that a reasonable loss distribution is observed for the average client.

A 2-level hierarchical, random topology was used, where every node in a random first-level graph is connected to a separate random second level graph by a single link. All second-level graphs have the same number of nodes, and each node in a second-level graph has either a client or the one source attached to it by an additional link.

4.2 Generating Comparable Topologies

For simulation results using different topologies to be comparable, we must somehow make all of our network graphs comparable to the graphs used in the original STORM study (henceforth referred to as *2-level*). The following features must be taken into consideration: number of nodes; average out-degree; number of low/high latency links (relative sizes of the first and second levels); and the location of clients and the source (the *participants*). In addition, graphs generated by Waxman, PLOD, and Recursive are not necessarily connected: we remedy this by repeatedly finding and identifying disconnected components in the graph and randomly connecting the smallest to the largest until only one component remains.

The number of nodes is a parameter to Waxman, PLOD, and Recursive, but Recursive requires that the number of nodes is a power of two. For each experiment, we use the the value of N closest to the corresponding 2-level graph. Similarly, we aligned the average out-degree of all graphs.

In the 2-level networks, participants are linked to every second-level node. For the other three generation methods, we link participants to arbitrary nodes, but ensure that there is most one participant per node.

The first-level of the 2-level networks is always composed of high-latency links, while the second-level has low-latency links—we must ensure that this feature is modeled in the other topologies as well. The Recursive algorithm provides a distinction between high and low latency links, while Waxman and PLOD do not. For these other two algorithms, we devised an independent method of assigning low latency links, shown in Figure 11. This simple algorithm ensures that all links near participant nodes are low-latency, and assigns low-latency links randomly otherwise.

4.3 Re-Evaluation

We now present our re-evaluation of the STORM multicast scheme. The original study measured average packet overhead for a varying number of clients, as well as the loss distribution for a 100-client topology. We perform the same measurements, with the STORM simulator configured as shown in Table 2. For each of the topology generators, Figure 12 shows the ratio of protocol packets received per packet recovered for an increasing number of clients[4]. For this

[4] Our results for the 2-level topology are not identical to the original storm study for two reasons: the STORM simulator has since been modified to remove several bugs; and we average over five random runs where the original study did not.

$assign\_low\_lat\_links(M_l) =$

Let M_l be the desired number of low-latency edges

FOREACH edge e in the graph
  mark e as being high-latency

FOREACH node n linked to a participant
  if there is a high-latency edge e out of n
    mark e as being low-latency
    $M_l --$

WHILE $M_l \geq 0$
  mark any remaining high-latency edge as being low-latency
  $M_l --$

Figure 11: Algorithm for assigning low-latency edges

Table 2: STORM Simulation Parameters

Parameter	Value
high-latency link delays	20-40 ms, uniform random
low-latency link delays	1-5 ms, uniform random
loss rates	0.1 to 0.5 %, uniform random
playback buffer sizes	500ms
simulation duration	10 minutes of simulated time

experiment, we average over 5 random topologies for each topology generator. We see that the STORM scheme scales well for all topologies: the overhead ratio is consistently near three, with the exception of the results for 10-client topologies which are skewed because the small graphs produce results with very high variance. The breakdown of overhead into *nacks* and *repairs* is more or less constant, each accounting for roughly half of the overhead. However, these average results do not tell the whole story.

Our intuition is that power-law topologies should produce the same average results as the corresponding non-power-law topologies, but that the distributions for a given measurement will vary significantly by topology. For each topology generator, Figure 13 shows the percent of protocol overhead incurred per node in the network. The 2-level generator used in the original STORM study has a vastly different overhead distribution than the other generators. Evidently, the distributions for the other generators follow a power-law, since their trends are nearly linear in log-log space. We repeat that even a small deviation in log-log space translates to a large deviation in normal space—so even the Waxman topologies give significantly different distributions than the two power-law topologies. The trend for the 2-level distribution appears

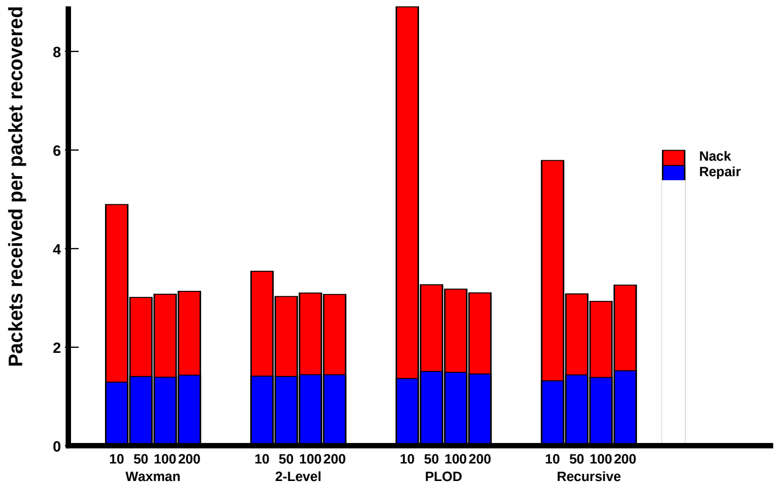


Figure 12: STORM protocol overhead for varying numbers of clients.

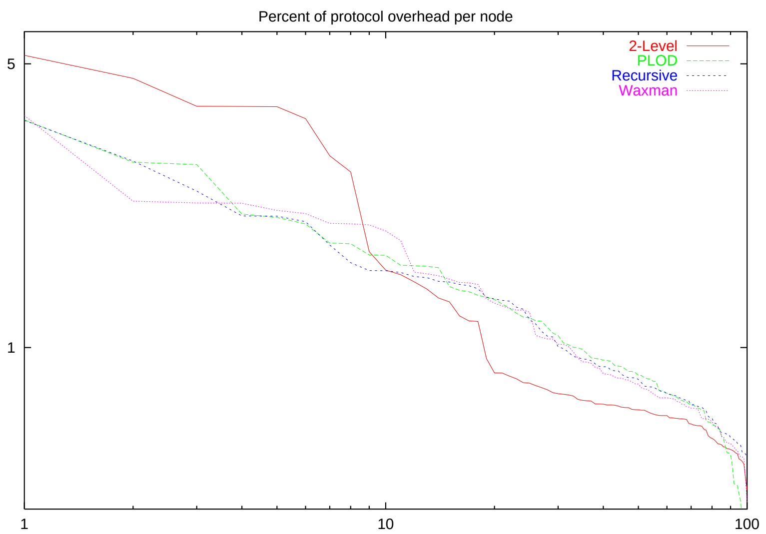


Figure 13: Percent of protocol overhead per node for 100-client topologies.

to have two distinct regions—these are likely related to this topology's two distinct levels. The other generators produce a more smooth distribution, indicating that their topologies are more "natural".

Finally, we measure the distribution of lost packets over all nodes, as shown in Figure 14. Again, we see that topology has a large impact on the resulting loss distributions. Note that we are not advocating that a given topology produces more accurate results, but that the resulting distributions are evidently closely tied to the underlying topology. This suggests that selection of a topology generator is very important when performing a study which focuses on the distribution of a given metric. In most cases, a good solution may be to repeat the

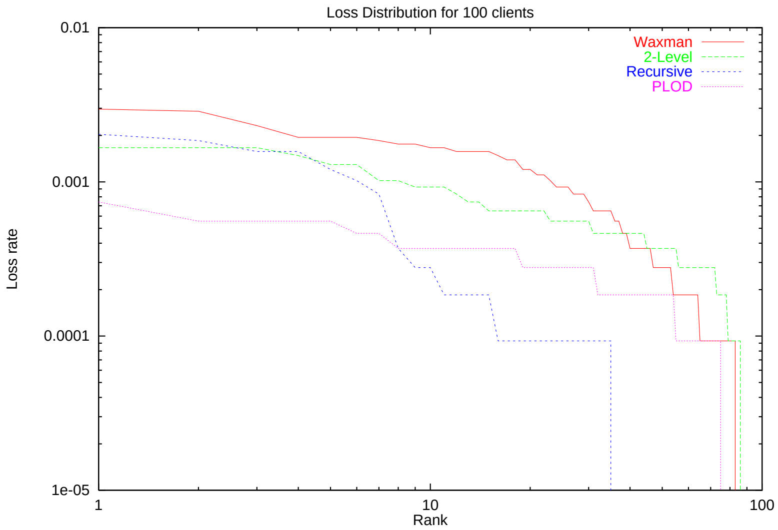


Figure 14: Loss distribution of 100-client topologies.

experiment using a variety of topology generators.

5 Conclusions

Recent work has shown that power-laws govern the topologies of real internets, particularly the Internet. We have shown that topology generators currently used in simulated network studies produce topologies that do not follow all of these power-laws. We also presented two methods for generating topologies that do obey all four power-laws. To show that topology is an important consideration in simulated network experiments, we re-evaluated the STORM study on several topologies, including our new power-law topologies. Results showed that average metrics are relatively unaffected by topology, but that distributions vary significantly. This suggests that if the distribution of a metric is important to a study that topology must be carefully chosen—a good strategy is to use multiple topology generators, including those that obey power-laws.

References

[1] R. Albert, H. Jeong, and A.-L. Barabási. The diameter of the world wide web. Submitted and available at `http://www.nd.edu/~alb/public.html`.

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286, 1999.

[3] Eric Bauer. Private communication.

[4] K. L. Calvert, M. B. Doar, and E. W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, June 1997.

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, 1999.

[6] B. A. Huberman and L. A. Adamic. Evolutionary dynamics of the world wide web. Technical report, Xerox Palo Alto Research Center, February 1999.

[7] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: measurements, models and methods. In *Proceedings of the International Conference on Combinatorics and Computing*, 1999.

[8] S. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Extracting large scale knowledge bases from the web. In *IEEE International conference on Very Large Databases (VLDB)*, Edinburgh, Scotland, 1999.

[9] B. M. Waxman. Routing of multipoint connections. *IEEE J. Selected Areas in Communications 6(9)*, December 1988.

[10] X. R. Xu, A. C. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous-media applications. In *NOSSDAV*, 1997.

[11] E. Zegura, K. Calvert, and M. Donahoo. A quantitative comparison of graph-based models for internet topology. *Transactions on Networking*, December 1997.