Generating Network Topologies That Obey Power Laws

Christopher R. Palmer
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, 15213
crpalmer@cs.cmu.edu

J. Gregory Steffan
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, 15213
steffan@cs.cmu.edu

Abstract— **Recent studies have shown that Internet graphs and other network systems follow power-laws. Are these laws obeyed by the artificial network topologies used in network simulations? Does it matter? 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. We also re-evaluate a multicast study to show the impact of using power-law topologies.**

I. Introduction

Recent studies have shown that Internet graphs follow power laws [1]—i.e., certain graph metrics follow the distribution $y \propto x^{\alpha}$. This phenomenon has been observed in router topology, inter-domain topology [1], and the world-wide-web [2], [3], [4], [5], [6]. 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 conclusions 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 [7] 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 use them to evaluate current topology generators. In Section III, we present and evaluate two algorithms for generating power-law topologies, and in Section IV we explore the impact of power-law topologies on the STORM multicast scheme. Finally, we conclude in Section V.

II. Power Laws and Topology Generators

In this Section, we describe the power-laws defined by Faloutsos *et al* [1]. Then, we explain the commonly used topology generators. Finally, we show that these topology generators do not follow all of the power-laws.

A. *The Faloutsos* et al. *Power-Laws*

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

Power-Law 1: rank exponent $\mathcal{R}$ 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 $\mathcal{O}$ 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 $\mathcal{H}$ 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, $\mathcal{H}$: $P(h) \propto h^{\mathcal{H}}, h \ll \delta$, the diameter.

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

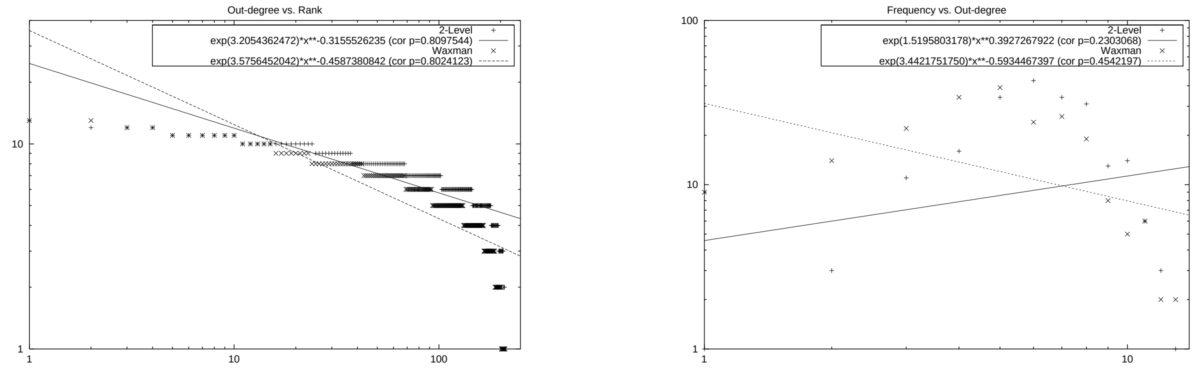
Faloutsos *et al* suggest using these power-laws to measure the realism of synthetic graphs [1].

B. *Current Topology Generators*

Topology generators currently used in simulation studies are constructed to obtain a desired average out-degree, and may even produce a strict 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 [8] generator 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

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



a) Out-degree vs Rank b) Frequency vs out-degree

Fig. 1. Power-law plots for current topology generators.

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 [9], [10] 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.

C. Performance on Power-Laws

A power-law defines 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 200 node networks are generated using Waxman's algorithm and the 2-level algorithm used in the STORM [7] study. The *rank exponent* and *out-degree exponent* graphs for these generators appear in Fig. 1 while the *hop-plot exponent* and *eigen exponent* plots have been elided to save space. The hopcounts and the eigen values graphs show strong power-law relationships, while the degree based graphs (Fig. 1a and 1b) show no power law relationships. In particular, the out-degree vs. rank graph is definitely concave and not linear ($\rho = .8$). The frequency vs. out-degree graph appears to define a bell-curve in log-log space ($\rho < .5$). To summarize, neither Waxman's algorithm nor the 2-level hierarchical algorithm generate graphs that pass all of the power-law tests.

III. New Generators for Power-Law Topologies

Here 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.

A. 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 $A_{u,v}$. Our algorithms assume that we want to generate M edges.

B. 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 Fig. 2. First, we randomly assign to each node a number of *out-degree credits* using the exponential distribution

$$\beta x^{-\alpha}$$

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 credits 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 credits 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.

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 (where $x = 1$, for the node with largest out-degree). Once we have chosen β, we select a value of α that produces the desired number of edges.

```
PLOD(N, M, A) =
    FOR i : 1..N
        x = uniform_random(1, N)
        degree_i = βx^(-α)
    FOR i : 1..M
        WHILE 1
            r = uniform_random(1, N)
            c = uniform_random(1, N)
            IF r ≠ c AND degree_r AND degree_c AND !A_{r,c}
                degree_r--, degree_c--
                A_{r,c} = 1, A_{c,r} = 1
                BREAK
```

Fig. 2. Power-law out-degree generator

C. Recursive Topology Generator

The recursive topology generator has two components. First, we define a probability distribution function that randomly selects a pair of nodes. Second, we use this 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 (adjacency matrix) we defined the quantities

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

with $\alpha + \beta + \gamma + \epsilon = 1$. These values correspond to the percentage of edges we want to place in each recursive subarray. The generator must take care to produce a symmetric matrix with zero diagonal (bidirectional links and no self-links in the network). 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. 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).

Let N_1 be the number of potentially non-zero values in the upper-left/lower-right quadrants of the matrix and N_2 the number of potentially non-zero values in the top-left/bottom-right quadrants. Define

$$p_\alpha \propto \frac{\alpha}{N_1} \qquad p_{\gamma,\beta} \propto \frac{\beta+\gamma}{N_2} \qquad p_\epsilon \propto \frac{\epsilon}{N_1}$$

to be the probability of a link in each quadrant.

Fig. 3 provides an algorithm, *gensym*, that will randomly select an edge, (u, v), using the probability distribution function we have defined. The selected edge will always be in the upper triangle.

Our topology generator has parameters N and M (the number of nodes and edges, respectively) and the probabilities α, β, γ and ϵ. It then generates a weighted graph using the edge generator by repeatedly selecting a single link and incrementing the weight of that link until such time as we have M links. The details are given in Fig. 3.

It is up to the user to decide how to map the weightings to network capabilities. For example, the experimental results presented in the following section have two classes of links: high latency and low latency. We have defined high latency links to be the more heavily weighted ones (assuming that weight corresponds to some measure of the traffic passing through a link).

```
gensym(n) =
    Let N_1 = (n/2)^2 - (n/2)
    Let N_2 = (n/2)^2
    Let κ = 1/( (1/N_1)·α + (1/N_2)(β + γ) + (1/N_1)ε )
    Let A = [ A_11  A_12 ]
            [ A_21  A_22 ]
    IF n = 2 THEN return (1, 2)
    ELSE WITH PROBABILITY
        α · κ/N_1          ⇒ RETURN gensym(n/2)
        (β + γ) · κ/N_2    ⇒ RETURN gen(n/2) + (0, n/2)
        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)

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
```

Fig. 3. Recursive topology generator

D. Evaluation

We evaluated PLOD and Recursive on the four power-laws, as shown in Fig. 4. PLOD has a nearly perfect out-degree vs. rank plot while Recursive appears as a characteristic 80-20 curve (Fig. 4a). All other metrics (Fig. 4b-d) exhibit excellent power-law relationships ($\rho \geq .92$). We conclude that PLOD and Recursive capture Internet characteristics far better than either Waxman or 2-level.

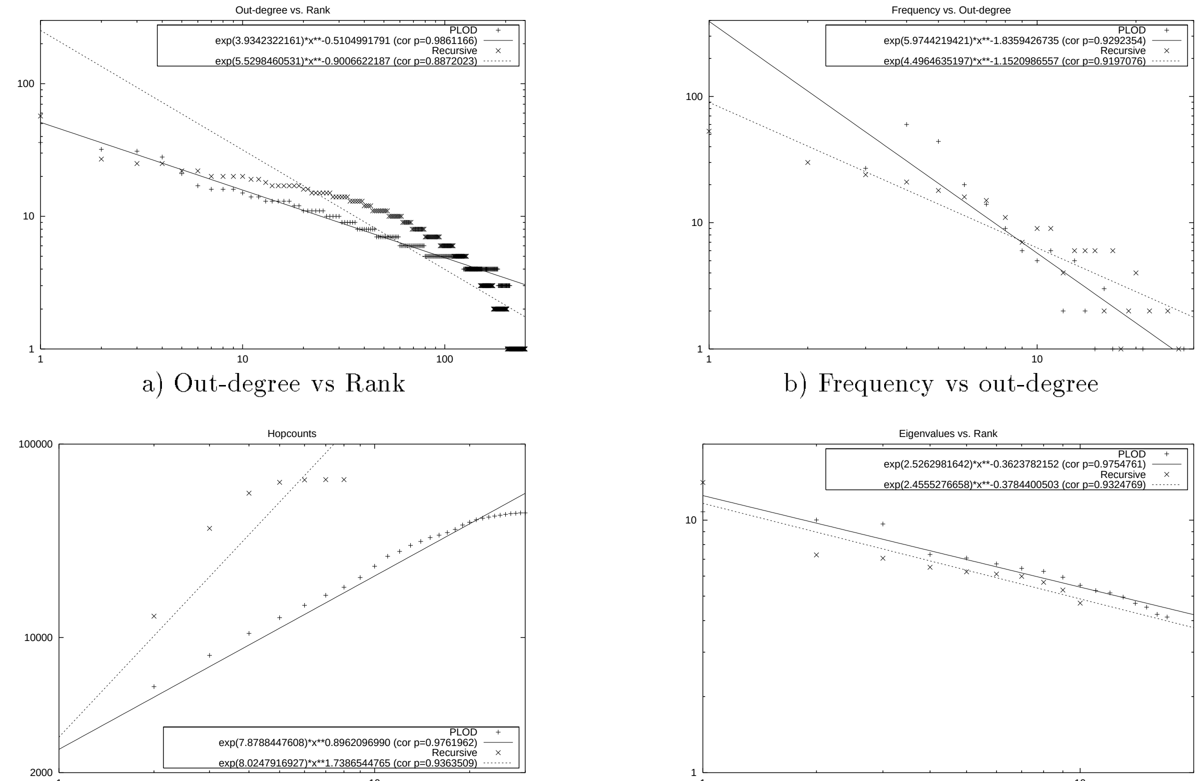
IV. Impact of Power-Law Topologies

In this section, we investigate the impact of power-law topologies on simulation results. In particular, we re-evaluate the STORM [7] 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. The STORM Multicast Scheme

The STORM multicast algorithm [7] provides a scalable method of recovering lost multicast packets. Each receiver (*client*) chooses a *parent* node from a set of neighboring multicast participants. When a packet is late, the client sends its 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 scaled as the number of clients in-



a) Out-degree vs Rank

b) Frequency vs out-degree

c) Hopcounts

d) Eigenvalues vs rank

Fig. 4. Power-law plots for our new topology generators.

creased, and that the loss-distribution observed for the average client was reasonable. A random 2-level hierarchical 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 is attached to either a client or to the source by an additional link.

B. Generating Comparable Topologies

For simulation results using different topologies to be comparable, we must make all of our network graphs comparable to the graphs used in the original STORM study (henceforth referred to as *2-level*). The following five features must be taken into consideration.

First, the number of nodes must be comparable. This is a parameter to Waxman, PLOD, and Recursive, but Recursive requires that the number of nodes be a power of two. For each experiment, we use the value of N closest to the corresponding 2-level graph. Second, we aligned the average out-degree of all graphs. Third, in the 2-level networks, multicast participants are linked to every second-level node, while for the other three generation methods we link participants to arbitrary nodes and ensure that there is at most one participant per node. Fourth, the first-level of a 2-level network 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 that ensures that all links near participant nodes are low-latency, and assigns low-latency links randomly otherwise. Finally, 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.

C. 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. For each of the topology generators, Fig. 5 shows the ratio of protocol packets received per packet recovered for an increasing number of clients. For every 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

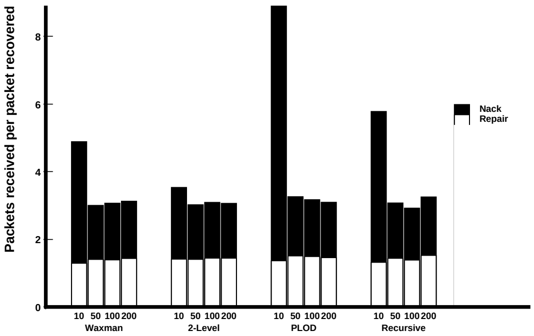


Fig. 5. STORM protocol overhead for varying numbers of clients.

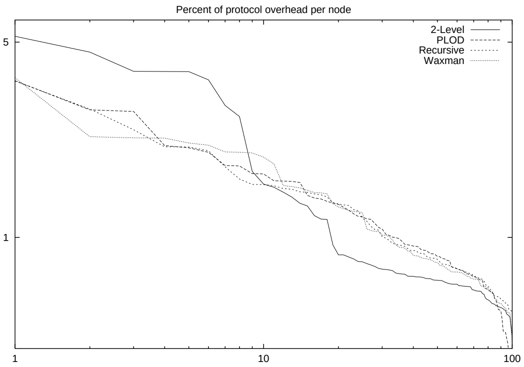


Fig. 6. Percent of protocol overhead per node for 100 clients.

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, Fig. 6 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. The protocol overheads for the 2-level distribution appear to have two distinct regions—these are likely related to this topology's two distinct levels. The other generators produce a more smooth distribution, perhaps indicating that their topologies are more "natural."

Finally, we measure the distribution of lost packets over all nodes, as shown in Fig. 7. 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 experiment using a variety of topology generators.

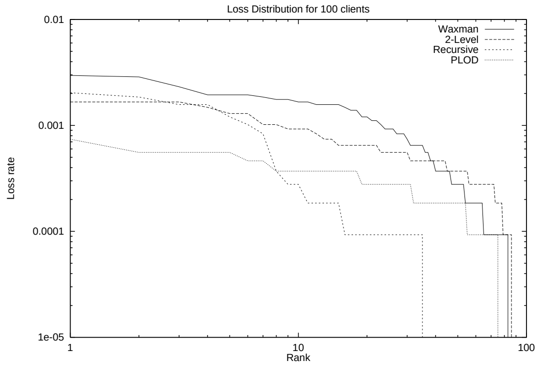


Fig. 7. Loss distribution for 100 clients.

V. Conclusions

Recent work has shown that power-laws govern the topologies of real inter-networks, particularly the Internet. This work and other work done concurrently [11] show that the topology generators currently used for simulation studies produce topologies that do not follow all of these power-laws, and present new 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 show 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 your conclusion, that topology must be chosen carefully—a good strategy is to use multiple topology generators, including those that obey power-laws.

References

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

[2] 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`.

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

[4] B. A. Huberman and L. A. Adamic, "Evolutionary dynamics of the world wide web," Tech. Rep., Xerox Palo Alto Research Center, February 1999.

[5] 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.

[6] 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.

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

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

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

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

[11] A. Medina, I. Matta, and J. Byers, "On the origin of power laws in internet topologies," Tech. Rep., Boston University, January 2000.