The Different Dimensions of Dynamicity

Mayur Deshpande and Nalini Venkatasubramanian
Donald Bren School of Information and Computer Science
University of California, Irvine
Email: {mayur, nalini}@ics.uci.edu

Abstract

In a Peer-to-Peer (P2P) network, a 'fabric' of overlay links helps peers discover and use other peers' resources. This fabric, however, is highly dynamic and constantly changing. While different measures of dynamicity have been implicitly and explicitly proposed, there is lack of deeper understanding about the various aspects of dynamicity. In this paper we systematically evaluate and quantify different dimensions of dynamicity through controlled generation of different dynamic networks. We also introduce a new dimension of dynamicity, persistence*, which quantifies stable nodes in a network. This measure could be quite useful in the design and testing of P2P protocols that exploit the presence of stable nodes. Also, quite coincidentally, and to our surprise, a certain type of dynamic network that we designed has node degree properties that resemble those observed in social networks.*

1. Introduction and Related Work

Peer-to-Peer(P2P) networks are evolving into an important paradigm to support scalable, fault-tolerant and low cost sharing of information and resources over the internet. There is, therefore, a growing and widespread interest in researching P2P networks to make them more effective and efficient. Researchers have explored [3] use of novel data-structure abstractions such as Distributed Hash Table [15] to construct overlays that have theoretical guarantees for object location and routing. Others propose making current P2P application protocols more efficient through better search protocols and scalable architectures [6, 17].

Many have also been actively observing, measuring and quantifying different characteristics of current P2P systems [16, 9, 5] to get a 'grounds-up' understanding of user/peer behavior in these networks. The studies show that a key characteristic of P2P networks is their highly dynamic nature [16]. In this respect, P2P networks are quite different from other (relatively) well studied large networks such as the Internet router network (routers are designed explicitly to be fail-safe) or the World Wide Web (WWW) (where we-pages exist from weeks to years). Furthermore, understanding basic characteristics of a network impacts the protocols designed to operate in that environment [14]. We believe that, for P2P networks, understanding overlay dynamicity can be a key factor in designing effective and efficient protocols and architectures.

1.1. Related Work

Dynamicity measure of files/objects in a P2P network have been well studied and documented [9]. Empirical data on aggregate node statistics (such as uptime) have also been documented [16]. However, there seem to be a lack of statistics that can reveal in a straightforward and intuitive way the different dynamic characteristics of the whole network of nodes.

Previous work [11, 5] has has addressed this issue to a certain extent. In [11], the authors propose a measure of dynamicity called *Half-life* which measures the time taken for half of the nodes in the network to change. Thus, a lower half-life directly corresponds to a more dynamic network. In [5], the authors (implicitly) define a notion of dynamicity called *turnover* which indicates how many nodes are joining and leaving the network in a given unit of time. However, these measures of dynamicity are not studied exclusively by themselves but rather are introduced to provide a basis to study or prove other properties of a P2P network. Moreover, there is no information on how these different metrics 'interact' in a network. For example, can one say that if a network has a high turnover it will have a small half-life?

1.2. Our Contribution

This paper makes three contributions to the study of network dynamicity:

1. In addition to half-life and turnover, we identify two more basic measures of network dynamcity, viz., *size-*

change and *persistence*. Persistence can be an useful metric to quantify stable nodes in a network.

2. We generate different dynamic networks systematically and measure the values of the four metrics in the networks. These measurements shed light on the correlations and behavior of the four metrics and show that no one metric is enough to characterize the dynamicity of the network by itself.

3. We study the 'evolution' of the different networks and observe that certain real-world network structures can in fact be explained using simple rules and processes.

The rest of the paper is as follows. In Section- 2 we present our model of a dynamic graph, which is used as a basis to define the various dynamicity measures. In Section- 3, we present our approach to generate different dynamic graphs. The dynamicity measures for these different graphs and the inter-relation between the different measures is explored in Section- 4. In Section- 5, we explore the distribution of degree of the emergent network for the different dynamic graphs. We conclude in Section- 6.

2. Dynamic Graph Model

In this section, we formally define a model of a dynamic graph and define various dynamicity measures with respect to this model.

Dynamic graphs have an implicit notion of change and time i.e the structure of the graph changes over time. An intuitive way to think about a dynamic graph is, as a 'recording' of snapshots of the evolving graph. Operations that took place between graph snapshots are recorded so that application of the operations on a previous snapshot yields the next snapshot. We define this notion more formally as follows:

A *Static Graph* is defined as a graph $G = \{V, E\}$ of vertices and edges. A *Dynamic Operation* on a static graph is any operation that changes the *structure* of the graph. We define four dynamic operations in our model: *AddVertex, DeleteVertex, AddEdge & DeleteEdge*. As their name suggests, the operations are quite straightforward. For example, *AddVertex* adds a new vertex to a graph.

Application of an ordered set of operations, $\{O_i\}$, on a static graph, G_i, leads to the *evolution* of the graph to G_{i+1}. G_i and G_{i+1} are called consecutive *Graph Snapshots*. Graph snapshots could also be time-stamped to denote the time at which the snapshots were taken. A graph snapshot is then denoted as G_i^t, the ith snapshot taken at time t.

A *Dynamic Graph* D^G is an ordered set of graph snapshots and the set of operations following each snapshot:

$$D^G = \{(G_r^0, O_r^0), (G_s^1, O_s^1)..(G_t^i, O_t^i)\},\ r < s < t \quad (1)$$

The set of operations, O_i is never null i.e. two consecutive snapshots differ by at least one vertex or one edge. The vertex set and edge set of a graph snapshot G_i^t are denoted as V_i^t and E_i^t respectively. Sometimes, for simplicity, a graph snapshot is referenced by either just its snapshot number as G_i or the time of the snapshot as G^t, as is appropriate. Also, $G_i \cap G_{i+1}$ may include the null set ($\{\emptyset\}$) i.e. the set of vertices, if any, in G_{i+1} are completely different from the ones in G_i.

Using this model of the dynamic graph in, we now define various dynamicity measures. These measures are all defined exclusively with respect to vertices. Analogous measures with respect to edges could also be defined.

2.1. Dynamicity Measures

2.1.1. Graph Turnover We define graph turnover, θ, as the fraction of vertices that have 'left' the graph between two snapshots over the total number of vertices in the initial snapshot:

$$\theta(G_i, O_i, G_j,) = NumDeleteVertex(O_i)/|V_i| \quad (2)$$

$NumDeleteVertex(O_i)$ returns the number of *DeleteVertex* operations in O_i. O_i is the set of operations needed to transition G_i to G_j.

Turnover could also be measured as the total number of vertices added ($AddVertex$) during a particular timeframe or the combination of both $AddVertex$ and $DeleteVertex$ operations. Turnover has been mentioned implicitly in [5].

2.1.2. Half-life Half-life of a graph is the minimum time taken for half of the vertices in a graph to change (either through addition or deletion). It can be defined (for a given time of reference, i) as:

$$HalfLife(D^G, i) = (j - i) \wedge min(j) \wedge j > i \geq t \quad (3)$$
$$|V^i \cap V^j| \geq |V^i|/2 \text{ or } |V^i \cap V^j| \geq |V^j|/2$$

Therefore, half-life is the minimum time needed so that the two snapshots differ by at least half of the nodes in either snapshot (see [11] for a more detailed explanation and analysis).

2.1.3. Graph Size-Change Graph size-change, δ, given a graph snapshot, G_i, and the set of operations, O, performed on G_i is defined as:

$$\delta(G_i, O) = 1 - MinSize(G_i, O)/MaxSize(G_i, O) \quad (4)$$

Let $(g_1, g_2, ...g_i)$ denote the graphs that have evolved from G_i after application of each operation in $\{O\}$ $(o_i, o_2, ...o_i)$. Note that g_i results after application of of all operations, from $(o_1...o_i)$. Using this, we define:

$$MinSize(G_i, O) = min(|v_1|, |v_2|, ...|v_i|)$$
$$MaxSize(G_i, O) = max(|v_1|, |v_2|, ...|v_i|)$$

Graph size-change, therefore, gives an intuitive feel into the *fluctuation* of the size of a graph (in vertices) between the two snapshots. A value close to zero indicates that the graph size did not vary by much, while a value close to one indicates the opposite.

2.1.4. Graph Persistence We define a measure for graph dynamicity called *Graph Persistence*, σ. Persistence between two snapshots G_i and G_j, $j > i$ (or even G^i and G^j), is defined as:

$$\sigma(G_i, G_j) = 1 - (|V_i \cap V_J|)/|V_i| \quad (5)$$

i.e. the fraction of vertices remaining in the new snapshot over the total number of vertices in the old snapshot. Persistence of a graph gives an intuition into how much a graph has 'changed' (or not changed) between snapshots. A value close to zero indicates the graph has changed dramatically between the two snapshots while a value closer to one indicates that the change is small.

3. Generation of Dynamic Networks

In this section we describe our process for generating different dynamic networks. The networks are generated by combining three policies in various ways (the policies and their values are shown in Figure 2) . The policies control the distribution of lifetimes of nodes, how new nodes attach and how nodes leave the network. The policies are 'plugged' into a network simulator to simulate various types of dynamic networks. We first describe the simulator followed by the three policies.

3.1. The Dynamic Network Simulator

Our simulator (Fig. 1) follows the general dynamic graph model presented in Section 2. Each simulation is driven by certain *simulation parameters* (values used in our simulations are listed in Fig. 2). Each simulation is started with a graph snapshot that has the specified number of vertices. This initial snapshot is built using only $AddVertex$ and $AddEdge$ till the required amount of vertices are met ($GraphSize$).

After snapshot-0 is built, the graph is allowed to change. Snapshot-0 is equivalent to G_0^0 (snapshot zero at time zero). Time is incremented in 'ticks'. Each vertex has a unique lifetime and depending upon the *Lifetime Policy* the lifetimes of vertices follow certain distributions.

At each time tick, the *uptime* of each vertex is increased by 1. If a vertex has outlived it's life (uptime is greater than it's intended lifetime), it is removed from the graph. As the graph is evolving snapshots are taken at regular intervals. A snapshot of the graph is taken whenever $GraphSize$ new vertices have been added to the graph. This also implies that $GraphSize$ vertices have been removed from the graph since the last snapshot. All vertices that have been removed are recorded as well. When $SimulationVertices$ vertices have been generated, the simulation is stopped and a final snapshot is taken.

```
Dynamic Network Simulator
 Time T = 0
 SnapshotCount = 0
 GraphSnapshotList G(SnapshotCount) = G_0
 TotalVertices = 0
 LastCountVertices = 0
 GraphSize = SimulationParameter
 SimulationVertices = SimulationParameter
 LifetimePolicyLP = SimulationParameter
 AttachmentPolicyAP = SimulationParameter
 DetachmentPolicyDP = SimulationParameter

 DeadNodes = 0
 while (TotalVertices ≥ SimulationVertices)
     if (TotalVertices ≥ LastCountVertices + GraphSize)
         // Take Snapshot of Graph
         G(SnapshotCount) = G_T
         Increment SnapshotCount by 1
         LastCountVertices = TotalVertices

     ∀v ∈ {V_T} :
         Increment v_uptime by 1
         if (v_uptime = v_life)
             RemoveVertex(G_T, v)
             if (DP = Single)
                 v_n = GenerateNewVertex(LP)
                 AddVertex(G_T, v_n)
                 AddEdge(G_T, v_n) using AP
             ElseIf (DP = Aggregate)
                 Increment DeadNodes by 1
             Increment TotalVertices by 1
     If (DP = Aggregate)
         For DeadNodes times:
             v_n = GenerateNewVertex(LP)
             AddVertex(G_T, v_n)
             AddEdge(G_T, v_n) using AP

     Increment T by 1
```

Figure 1. Simulation Engine

3.2. Lifetime Policy

The lifetime policy decides from which distribution to pick a node's lifetime value. We use four distributions; three distributions are "off the self" and one is "custom" which we have explicitly designed. Lifetimes of nodes are picked at random from the distributions.

Parameter Name	Value(s)
$SimualtionVertices$	$1,000,000$
$GraphSize$	$100,000$
$LifetimeParameter$	100
$LifetimePolicy$	Uniform-Random, Gaussian, Custom, Power-Law
$AttachmentPolicy$	Random, Preferential
$DetachmentPolicy$	Single, Aggregate

Figure 2. Table of Simulation Parameters

3.2.1. Uniform-Random Generates uniform-random numbers from $1 - LifetimeParameter$. The Java Math.random method generates numbers of this type. This generator is referred to as the Uniform generator for the rest of the paper.

3.2.2. Gaussian Generates random numbers that follow a Gaussian-Normal distribution. We used the Java Gaussian generator for this purpose. This generator generates numbers between -1 and +1 with a mean of 0 and a variance of 1. We used the following method to change the mean and make all the generated numbers positive: $(p(x) + 1.0) * LifetimeParameter$, where $p(x)$ is the number generated from the Java gaussian generator.

3.2.3. Power-Law Generates random numbers that follow a Power-Law distribution. Power-Laws are characterized by the following formula:

$$f(x) = \beta x^{\alpha} \tag{6}$$

In our simulations, β is the maximum lifetime of a vertex ($LifetimeParameter$). x was chosen uniformly at random from from the range $[1..SimulationVertices]$.

3.2.4. Custom We devised a "mixture" generator that mixes seven different generators: one a power-law generator and six other gaussian generators with different means. This generator is labeled as 'Custom' in all figures. The algorithm used by this generator is described in Figure 3. Our intuition for this custom generator comes from the fact that observed uptimes of peers do not follow any of the 'off-the-shelf' distributions in a straightforward way. However, it may be possible that different groups of peers in a P2P network follow different distributions. The custom distribution also came closest (of all the four distributions) to modeling the distribution that was reported in Figure8(b) of [16].

We plotted the distributions of the lifetimes generated by the different generators (Figure 4) (we generated 10 million values for each distribution). as a Cumulative Density Function (CDF) of probabilities, i.e. the probability that a generated lifetime is greater than a certain value. This figure is useful to illustrate the presence of a power-law in a

Custom Lifetime Generator

$GraphSize$ = SimulationParameter
$MaxLifetime$ = LifeTimeParameter

$luck$ = Random () /* Between 0.0 - 1.0 */
if ($luck > 0.99$) return Gaussian() * MaxLifetime/4
if ($luck > 0.95$) return Gaussian() * MaxLifetime/8
if ($luck > 0.9$) return Gaussian() * MaxLifetime/16
if ($luck > 0.8$) return Gaussian() * MaxLifetime/30
if ($luck > 0.6$) return Gaussian() * MaxLifetime/50
if ($luck > 0.5$) return Gaussian() * MaxLifetime/75
if ($luck > 0.4$) return PowerLaw()

Figure 3. The Custom Lifetime Generator

straightforward manner (both axes are log-log scale). However, as can be seen in Figure 4, the power-law generator does not yield a "straight line" in the log-log plot. The imprecision may be due to rounding errors in floating point operations.

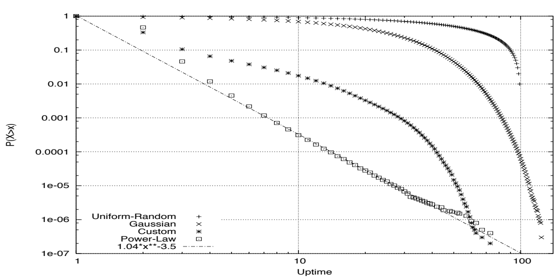


Figure 4. Vertex Lifetimes Distribution as Probability CDF

3.3. Attachment Policy

When new nodes join the network, they do so by forming an edge to one of the existing nodes in the network. How they form this edge is dictated by the *attachment policy*. An attachment policy is defined as a stochastic rule. We use two attachment policies in our network generation: *Random* and *Preferential*. These are defined as follows (v is the new node and v_i is an existing node in the graph):

$$Random : p(v, v_i) = 1/|V| \tag{7}$$

$$Preferential : p(v, v_i) = degree(v_i)/|E| \tag{8}$$

The random policy is quite straightforward to implement. A node picks an existing node at random to join. The preferential attachment algorithm is more sophisticated and is borrowed from [12].

3.4. Detachment Policy

In our simulator, we also control how nodes leave the network. This is called the *Detachment Policy*. In *Single Detachment Policy*, nodes leave 'one at a time', i.e., a new node is generated immediately after a node leaves the network. Thus, at most, the size of the network changes by 1. In *Aggregate Detachment Policy*, all nodes that have outlived their life (at a particular time tick) are removed together. New nodes equal in number to the removed nodes are then generated and added back to the network.

4. Empirical Comparison of Dynamicity Measures

In this section we analyze the various dynamic networks (Section 3) using the different dynamicity metrics (Section 2) and comment on the appropriateness of the dynamicity metrics to measure a particular aspect of dynamicity.

4.1. Turnover

In our network simulator, we simulate a certain total number of nodes ($SimulationVertices$) and take snapshots of the dynamic network after every $GraphSize$ of nodes leave the network (Figure 1). When turnover is high, the time between snapshots is small (lesser simulation ticks) as compared to when the turnover is lower (higher simulation ticks). Figure 5(a) shows this.

Of interest is the fact that Custom has a higher turnover than Power-Law (one would have expected the opposite). The reason for this is becasue 68% of nodes in Custom have lifetimes less than 1 as compared to 50% in Power-Law.

4.2. Persistence

Persistence gives an intuition into the presence of stable nodes in a network. Figure 5(b) shows the persistence value for the different lifetime policies across snapshots. We measure persistence after a fixed number of nodes have left the network and not in relation to time ticks. The reason for this is to make the persistence statistic independent of time frame and compare the different networks in a objective and straightforward way (Note that the average lifetimes for the different lifetime policies are different).

The results in Figure 5(b) are counter-intuitive if turnover and persistence are expected to be directly correlated. If they were, Uniform should have higher persistence than Gaussian, and Power-law higher than Custom. Surprisingly, Custom has a higher persistence than both Power-Law and Uniform! Thus, in the Custom-generated network, more nodes are persistent even though a higher number of nodes have lesser lifetime. A key motivation in the design of the Custom generator was also to illustrate this counter-intuition. In P2P networks, it is important to differentiate between the more stable in the network (some protocols such as KaZaa already do this by making stability as one of the parameters in the selection of super-nodes). Doing so, one can utilize the more stable nodes to make the P2P protocol, as a whole, more scalable.

4.3. Half-life

Half-life measures the time it takes for half of the network to change. From a given snapshot, we measure the earliest tick when half of the network has changed. Figure 5(c) plots the half-life when different lifetime policies are used.

As can be seen from Figure 5(c), the correlation between turnover (Figure 5(a)) and half-life is quite apparent; the lower the turnover, the higher the half-life (For custom and power-law, our granularity of measurements in time ticks was not sufficient to reveal the differences between them. Since their turnovers are greater than 25% for each time tick, their half-life is also 1 time-tick).

4.4. Size-change

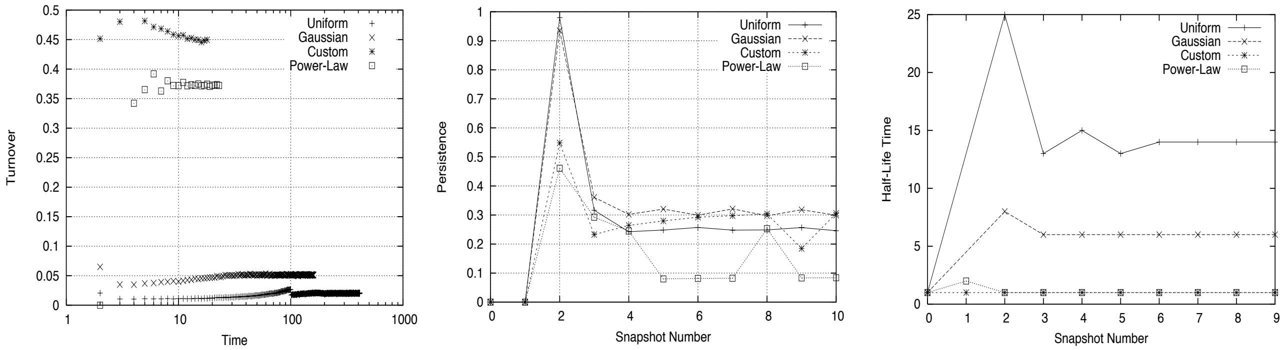
Size-change measures how much the network as a whole has fluctuated in size. In single-change Detachment Policy (DP), the size change of the network is the same (difference of 1 node at maximum). In aggregate DP, however, the more nodes that, the bigger the size change. However, given the way, size-change is defined, in our simulation model, turnover and size-change are identical for aggregate DP. To understand this, consider the definition of size-change. Starting with the definition of size change 4, $MinSize$ is equivalent to ($GraphSize - DeadNodes$). ($DeadNodes$ are the total number of nodes that died during a time tick). $MaxSize$ is $GraphSize$. Therefore,

$$\begin{aligned}\delta &= 1 - ((GraphSize - DeadNodes)/GraphSize)\\ &= (DeadNodes/GraphSize)\\ &= Turnover\end{aligned}$$

Size-change has an impact more on the evolution of the structure of the network and this is discussed in Section- 5.

Summary

Different dynamicity measures are useful in showing different characteristics of the network. Moreover, they may not be directly correlated. A higher turnover does not necessarily mean that the network is more 'volatile' in totality. Additionally, a combination of measures may give a more complete picture. For example, using both turnover



(a) Turnover (b) Persistence (c) Half-Life

Figure 5. Dynamicity Measures of Different Networks

and size-change we can judge how fast (turnover) a network is growing or shrinking (size-change), if at all.

5. Dynamicity Impact on Structure of Network

In this section we explore the impact of dynamicity on the structure of a network. By structure, we mean the distribution of edges (nodes degree) in the network. A power-law distribution, for example, indicates presence of hubs [4] while an exponential distribution is a characteristic of a 'Random Network' [8]. Our goal is to study whether dynamicity impacts the structure of a network and if so, what is the impact.

This could have important implications into the design of how nodes in a P2P network initiate and form connections with each other. As we'll describe, dynamicity combined with different attachment policies can lead to radically different network structures.

The effect of attachment policies in static or growing networks has been well studied across many types of actual networks and simulations [4, 1, 10, 7]. However, there has been relatively little research into how the structure changes as a network evolves for different kinds of dynamic networks. We feel that our work is (one of the) first to address this issue.

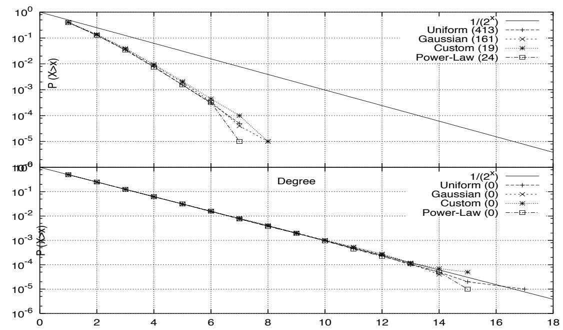
We study the impact of dynamicity on the structure of the network by plotting the degree distribution of the initial (G_0) and final (G_{10}) snapshot. The time tick of the final snapshot is indicated in parenthesis in the results. We compare the 'evolution' in the degree for all the combinations of Lifetime, Attachment and Detachment policies. The results are presented in the following sub sections. For brevity, networks using Random Attachment Policy are called as random-networks (and similarly preferential-networks. Additionally, we use the lifetime policy in the network name as well (for example, a uniform-random-network is a random-network in which the nodes follow uniform-random distribution in lifetimes).

5.1. Single Change Detachment Policy

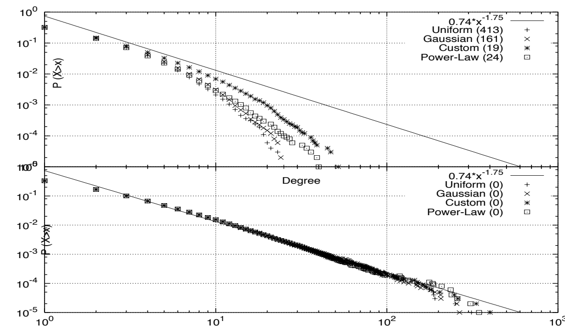
We study the impact of dynamicity when the detachment policy is single change. The results of the experiment are shown in Figure 6. The bottom part of the figures are snapshots from the first snapshot while the top parts are from the final snapshots (for the same attachment policy, we should not see any difference in the distribution of degree for the different lifetime policies at snapshot-0, since the lifetime policies have still to come into effect. However small differences can be seen at the tail of the distributions and these give an idea about the possible deviations in different runs of the simulator).

Analysis As a network evolves under the single change detachment policy, vertices with higher degree start to disappear and more vertices have lesser degree. This is observed in both random and preferential networks. The effect of Lifetime Policy (and hence different types of dynamicity) seems to have minimal effect on random-networks (the resulting distributions are still strongly exponential indicated by straight lines, though with a higher slope). The degree distributions for all the lifetime policies seem quite similar suggesting that a random-network is quite 'insensitive' to variation in lifetimes of vertices. Structured P2P networks (that use Distributed Hash Table) use a randomization function to select which node a new node should join to. In this regard, DHT-based P2P networks, can be thought of as random-networks. For these networks, therefore, dynamicity of nodes does not lead to any big change in the structure of the network (when size change is small).

In the case of preferential-networks, however, there is perceptible differences in the emergent distributions.; the



(a) Random Attachment Policy



(b) Preferential Attachment Policy

Figure 6. First and Last Snapshots of a Dynamic Graph Under Different Attachment and Lifetime Policies with Single Change Detachment Policy

power-laws are 'truncated' [13, 2]. Dynamicity of nodes combined with steady change leads to this truncation. Networks such as the WWW and router networks are predominantly growing (and show power-laws in their structure) whereas the networks we simulated are not growing but at a constant size. Networks where truncated power-laws have been observed are also approximately steady state (nodes join and leave but the rates of both are approximately equal). Examples of these are actor-networks (ties between actors who act together in movies) or social ecology networks (prey-predator ties in ecologies).

5.2. Aggregate Change Detachment Policy

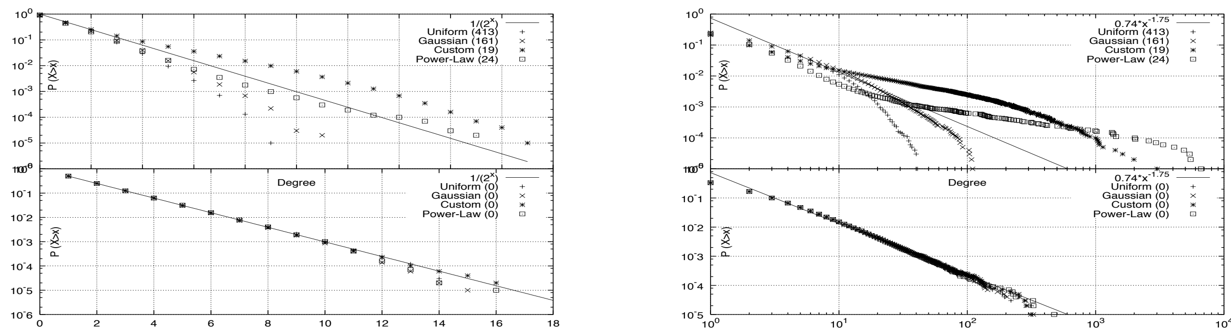
In aggregate detachment model, the difference in the size of the networks varies across the lifetime policy used; Custom has highest size difference while Uniform has the least (Figure 5(a)). Figure 7 shows the effect of dynamicity on the different networks when the aggregate change policy is used.

Analysis Unlike in single-change networks, aggregate-change networks (networks where aggregate change detachment policy is used) show both truncation and extension. In both random and preferential networks, power-law and custom-networks show distributions where vertices have higher degree (as compared to snapshot-0). In preferential-networks, this effect is highly pronounced with vertices having an orders of magnitude greater number of edges (Custom and Power-law). This is because, in aggregate change, a large percent of nodes are removed at once (in Custom and Power-law). Thus when new nodes join, they join nodes that already have many connections leading to those nodes becoming even more 'preferred'. As the network evolves, these preferred nodes amass a huge number of links to other nodes. Where the rate of leaving is not so high (Uniform and Gaussian), more nodes are available for new nodes to choose from. These networks, therefore, in contrast, exhibit truncation in degree distributions.

Also, it is interesting to see the effect of the interplay between persistence and size-change in the preferential attachment model. Custom has a higher size-change and higher persistence than power-law. Therefore more nodes leave and join the network at each time tick. However more nodes are also present consistently across time, for new nodes to join to. In Power-law, lesser number of nodes leave the network at each time tick but only a select few are able to stay in the network consistently. Therefore the select few end up acquiring massive number of edges as compared to the Custom-preferential-network (Figure 7(b)). For unstructured P2P networks with highly skewed lifetime distributions, therefore, protocols must be designed to avoid preferential attachment. Else, the load on 'preferred' nodes may increase exponentially making them a bottleneck to the whole system.

6. Conclusion and Future Work

In this paper, through exploratory simulation, we showed how systematically studying dynamicity may provide important clues and insights in understanding and modelling dynamic networks. However, it would be good to apply the results found here to real-world P2P networks. We are currently working on collecting data from real P2P applications and applying the dynamicity measures to characterize them. Many types of network dynamicity were not considered, such as network partitioning. Moreover, it is not obvious that the dynamic graph model that we used will be sufficient for all P2P networks (e.g. multicast addresses and caching other peers' addresses but not explicitly connecting to them are difficult to model using a graph). More sophisticated models (such as hyper-graphs) may be needed. However, this only shows that studying network dynamicity by itself is a rich area for exploration.



(a) Random Attachment Policy

(b) Preferential Attachment Policy

Figure 7. First and Last Snapshots of a Dynamic Graph Under Different Attachment and Lifetime Policies and with Aggregated Delete-And-Join Policy

Acknowledgements

We would like to thank Prof. Carter Butts and the four anonymous reviewers for their insightful comments and feedback. This work was supported by the National Science Foundation, under award numbers 0331707 and 0331690.

References

[1] R. Albert, H. Jeong, and A. L. Barabasi. The internet's achilles' heel: Error and attack tolerance of complex networks. *Nature*, 2000.

[2] L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley. Classes of small-world networks. *Proc. Natl. Acad. Sci. USA*, 97(21):11149–11152, 2000.

[3] H. Balakrishnan. Looking up data in p2p systems. In *Communications of the ACM (CACM)*, 2002.

[4] A.-L. Barabasi, R. Albert, and H. Jeong. Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A*, 281:69–77, 2000.

[5] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[6] A. Crespo and H. Garcia-Molina. Routing Indices For Peer-to-Peer Systems. In *Proceedings of the 22nd International Conference on Distributed Systems*, pages 23–32, Vienna, Austria, 2002.

[7] D. Eppstein and J. Wang. A steady state model for graph power laws. In *2nd International Workshop on Web Dynamics*, May 2002.

[8] P. Erdos and A. Renyi. On random graphs. *Publ. Math. Debrecen*, 6:290–297, 1959.

[9] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. In *Symposium on Operatinng Systems Principles (SOSP)*, October 2003.

[10] K. Klemm and V. M. Eguiluz. Highly clustered scale-free networks. *Physical Reeview E.*, 65, 2002.

[11] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *PODC*, pages 233–242, 2002.

[12] A. Medina, A. Lakhina, I. Matta, and J. Byers. Brite: Universal topology generation from a user's perspective. In *In Proceedings of Workshop the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS '01)*, October 2001.

[13] S. Mossa, M. Barthelemy, H. E. Stanley, and L. A. N. Amaral. Truncation of power law behavior in scale-free network models due to information filtering. *Physical Review Letters*, 88(13), 2002.

[14] C. R. Palmer and J. G. Steffan. Generating network topologies that obey power laws. In *In Proceedings of GLOBECOMM*, 2000.

[15] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Ninth annual ACM symposium on Parallel algorithms and architectures*, pages 311–320, 1997.

[16] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *ACM Transactions of Networking*, 12(2), 2004.

[17] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, March 2003.