社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－DPS－134 (5)
2008－CSEC－ 40 (5)
2008／3／6

P2Pシステムにおける動的ランダムオーバーレイの構築手法

呉 エキ　大下 福仁　角川 裕次　増澤 利光
大阪大学 情報科学研究科

抄録 — 本論文では、ランダムＰ２Ｐオーバーレイを動的なネットワーク上に構築する分散アルゴリズムを提案する。本研究では以下の二つの性質を有するオーバレイの近似的な構築を目的とする。（1）任意のノードに対し、システム中の他のノードが同じ確率でこのノードへの有向リンクを持つネットワークである。（2）各ノードがリンクされる確率はそのノードによる局所的な操作で調節可能である。本研究は、81種の候補アルゴリズムを出力する分散フレームワークを提案し、シミュレーションにより評価を行う。それらの内の二つの出力ネットワークは目的ネットワークに近い統計的な特徴を持つことをシミュレーションにより示す。

Building dynamic random peer-to-peer overlays

Yu Wu　Fukuhito Oosita　Hirotsugu Kakugawa　Tosimitsu Masuzawa
Graduate School of Information Science and Technology, Osaka University

Abstract — Random networks are applicable for P2P overlays because of robustness, balanced degree of edges and small world characteristics. A basic objective of this work is to build a network that each node v_i is adjacent from any other node by a given probability p_i. A node can locally adjust the probability to be linked. A distributed framework, which can output 81 kinds of different networks is presented and estimated by experimental approaches. Two outputs are proved have similar statistical aspects with our objective.

1 Introduction

1.1 Background

Peer-to-Peer(P2P) applications, with prominent advantages such as scalability, robustness and low investment requirement, have developed quickly in recent years. A P2P network is an overlay network over the IP-based Internet. The problem that how to construct and maintain an efficient P2P overlay attracts many attentions.

The Gnutella (version 0.4) network is the first pure P2P network with a decentralized overlay[1]. The Gnutella aimed to build a random overlay network. However, many studies show that the Gnutella overlay is far from a random network. It is a so-called power law distributed network. Compared to a random network of the same average degree, its diameter is larger and the topology is more planar. Such aspects worsen the performance of gossip-based information dissemination approaches, e.g. flooding and random walk. This is an important reason that Gnutella can not scale well, especially in resource searching.

Random networks are proved to have high connectivity, small diameters and be easy to be maintained by local link exchanging [2]. These features are particularly suitable for large-scale and highly dynamic P2P systems. Now, the idea of using a dynamic random network for P2P overlays is widely applied, e.g. by JXTA.

Normally, the random network means the uniform-random network in which each peer has a uniform probability to be linked by others. Such networks are designed to achieve load-balance

in gossip-based communication environments in which requests, e.g. search query and replication request, are delivered randomly. However, in a uniform-random network, informations (resources) are also distributed randomly in the system. That is the main reason of the difficulty of searching and management in P2P systems.

Some modern P2P systems, e.g. new Gnutella, KaZaA etc., artificially introduce some biases of degree in order to enhance the system performance by centrally managing informations in some peers of high degrees [3, 4]. Such systems often use a two-layered overlay which consists of an upper layer of some powerful peers, called super peer, and a lower layer of others. The upper layer is a connected sub-network. Peers in the lower layer are only linked to super peers in the upper layer. Most informations (resources) are concentrated to the upper layer and thus peers can achieve most of the system services by accessing neighboring supper peers.

1.2 Our contribution

An ideal P2P overlays are expected to have both the advantages of random networks and degree-free networks. On the one hand, the network shall be a dynamic random network that has a small diameter and high connectivity. On the other hand, each peer can have different degrees to improve the system performance.

The objective of this work is to build such a degree-controllable random network. An experimental approach is used to challenge this objective. We propose a framework in which a wide range of different overlay construction algorithms, some are existing and some are novel, are included. The framework, which is an extension of M. Jelasity's work, can output 81 kinds of different networks [5]. Simulation results prove that our framework can output better networks, both uniform-random and degree-controllable, than M. Jelasity's framework.

2 A Framework

2.1 Basic model

We model the network by a d-out-regular digraph $D(V,E)$ consists of n peers, $V = \{v_1, v_2, ..., v_n\}$. The network is a simple graph that self-loops and multiple links are invalid. A peer v_i can directly send messages to peer v_j if v_i has a directed link that points to v_j.

Each peer v_i has a local view, denoted by $view_i$, consists of at most d directed out-links. The links in the view are sorted by ascending order. Each link has an additional parameter called *hop count*. The hop count of a link from v_i to v_j is denoted by $H(e_{i,j})$, $e_{i,j} \in E$. Each peer v_i has a local parameter h_i, called initial hop count. The details of the parameter will be introduced later.

2.2 Detail description

The framework is a set of rules of how peers exchange links with each other. Each peer, denoted by v_i, periodically execute a 4-step processes including *(1) Target Selection, (2) Seed Planting, (3) View Merging, (4) View Selection.* Each step has three options. All peers in the system use a same option set. Detailed descriptions are as follows.

(1) Target selection: Peer v_i select an available link $e_{i,j}$ from its local view, denoted by $view_i$. The destination peer v_j of the link will be the target to exchange view with.

- Random: The link is uniform-randomly selected from $view_i$.
- Head: The first link in $view_i$ (with the lowest hop count) is selected.
- Tail: The last link in $view_i$ (with the highest hop count) is selected.

(2) Seed Planting: Some new links that point to v_i or v_j, called seeds, are created and inserted into the peers' view.

- Push: Peer v_i inserts its seed, denoted by $e_{j,i}$, into $view_j$[1]. The seed's hop count is the initial hop count of v_i, $H(e_{j,i}) = h_i$.
- Pull: Peer v_i inserts its seed, denoted by $e_{i,j}$, into $view_i$. The seed's hop count is the initial hop count of v_j, $H(e_{i,j}) = h_j$.
- Push&Pull: Executes both Push and Pull.

(3) View Merging: Peers v_i and v_j share their local views with each other.

- Push: Peer v_i increases the hop count of all links in $view_i$ by 1. Then v_i insert $view_i$ into $view_j$: $view_i \cup view_j \rightarrow view_j$.
- Pull: Peer v_j increases the hop count of all links in $view_j$ by 1. Then v_j insert $view_j$ into $view_i$: $view_i \cup view_j \rightarrow view_i$.
- Push&Pull: Executes both Push and Pull.

(4) View Selection: After the seed planting and view merging, some peers' views may include more than d links. The last step is to select d links to remain in a view.

- Random: Links are selected uniform-randomly from a view to remain.
- Head: The first d links, that have lower hop counts, in the view are selected to remain.
- Tail: The last d links, that have higher hop counts, in the view are selected to remain.

The framework includes $3^4 = 81$ combinations. Each of them outputs a different network. For simple, we denote each combinations by a 4-tuple (ts, sp, vm, vs) where ts, sp, vm, vs respectively indicates the options of target selection, seed planting, view merging and view selection. A wildcard is denoted by the symbol '*'. For example, $(random, push, *, head)$ indicates an output of random target selection, push seed planting, arbitrary view merging and head view selection.

[1]Notice $view_j$ may temporarily have more than d links.

The main difference of M. Jelasity's framework is that the seed planting and the view merging procedures are bounded to select a same option. Some outputs such as $(random/tail, push, push\&pull, head)$ are not available in it. However, that will be shown latter, only those two new outputs can achieve correct degree-control.

Outputs (*, *, *, *random/tail*) are excluded from further discussion. In the case of the random view selection, more in-links a peer has, its in-degree increases faster because each link has a same probability to be replicated. In the case of the tail view selection, if a link is replicated, it has a higher probability to be replicated next time. Thus, such networks result in highly skewed and uncontrollable topologies. The M. Jelasity's experimental results also agree with this standpoint. In the following of this paper, all discussions are based on the head view selection (*, *, *, *head*).

2.3 Degree control

Our basic objective is to build a network that each node v_i is adjacent from any other node by a given probability p_i. In implementation, the p_i can not be arbitrarily decided because the out-degree of every peer is at most d.

Our solution is to use a relative probability control mechanism. For any two peers v_i and v_j, the proportion p_i/p_j is expected to be $2^{h_j - h_i}$ where h_i, resp. h_j, is the initial hop count of p_i's, resp. p_j . A peer v_i can adjust its in-degree by just change its local parameter h_i.

The principle is as follows. In the framework, the hop count of any links can only be increased when they are replicated (merged into other peers' view). The only way to have a link with a small hop count is to create a seed with a initial hop count. In the case of head view selection, only links of low hop counts can remain. Thus, a peer has a low initial degree will have a higher in-degree. After a link of

Output	VAR	Sight	Connectedness	Max Diameter	Average Path Length
Uniform-random	30	956	Strong	3	2.35
Random, Push, Push	22	367	Strong	3	2.41
Random, Push, Pull	44	404	Strong	4	2.39
Random, Push, Push&Pull	48	702	Strong	4	2.41
Random, Pull, Push	**715**	282	**Weak**	-	-
Random, Pull, Pull	**1424**	282	**Weak**	-	-
Random, Pull, Push&Pull	**3016**	476	**Weak**	-	-
Random, Push&Pull, Push	70	358	Strong	4	2.43
Random, Push&Pull, Pull	71	379	Strong	4	2.41
Random, Push&Pull, Push&Pull	132	693	Strong	4	2.43
Head, Push, Push	25	**36**	Strong	3	2.38
Head, Push, Pull	43	**38**	Strong	3	2.36
Head, Push, Push&Pull	80	**46**	Strong	4	2.39
Head, Pull, Push	28	**30**	Strong	3	2.35
Head, Pull, Pull	**435**	**49**	Strong	4	2.48
Head, Pull, Push&Pull	**1300**	**79**	Strong	5	2.56
Head, Push, Push	28	**33**	Strong	3	2.36
Head, Push, Pull	54	**42**	Strong	3	2.37
Head, Push, Push&Pull	80	**47**	Strong	4	2.39
Tail, Push, Push	19	380	Strong	3	2.39
Tail, Push, Pull	41	421	Strong	4	2.38
Tail, Push, Push&Pull	47	700	Strong	4	2.41
Tail, Pull, Push	**736**	293	**Weak**	-	-
Tail, Pull, Pull	**927**	316	**Weak**	-	-
Tail, Pull, Push&Pull	**9065**	451	**Weak**	-	-
Tail, Push&Pull, Push	72	376	Strong	4	2.40
Tail, Push&Pull, Pull	74	403	Strong	4	2.39
Tail, Push&Pull, Push&Pull	144	701	Strong	4	2.42

Table 3.1: Main results of $(*,*,*,head)$ for uniform-random setting.
$n = 1000$, $d = 30$, 100 time cycles executed.

hop count h is replicated, two links, which point to a same peer, of hop count $h+1$ appears. Therefore, roughly, a peer has an initial hop count h has 2^k times higher probability to increase in-links than that of $h+k$.

2.4 Peer join and leave

When a peer v_i joins the system, we assumed that it can access at least one peer, called initiator, in the system[2]. Starting from the initiator, v_i randomly walks for several steps and reaches v_j. From $view_j$, v_i randomly selects a link $e_{j,k}$ and inserts it into $view_i$. Then, v_i starts the normal view exchanging procedures from a initial view $view_i = \{e_{i,k}\}, H(e_{i,k}) = H(e_{j,k})$.

When a peer leaves the system, no additional procedures are needed. Bad links which point to it can not remain for a long time because their hop counts can only be increased.

3 Simulation

We test the framework by simulation in order to find out some outputs agree with our objective. The simulation includes $n = 1000$ peers, the maximum out-degree of each peer is $d = 30$. In order not to introduce external random factors, peers execute in

[2]An initiator can be a initiator server or simply a website with some active peers' address.

a fixed order from v_1 to v_n in each time cycle[3].

3.1 Uniform degree setting

In the first simulation all peers' initial hop count are set to 0. In this case, the framework is expected to output uniform-random networks. We filter the outputs that have some statistical aspects obviously different from the uniform-random network or not suitable for P2P environments. The remained outputs can continue to be tested for degree control.

We introduce the following tests, which will be explained shortly, for graphs generated by our framework. Variance test of peers' in-degree, denoted by VAR, and dynamic view test, denoted by Sight. Main results are shown in Table 3.1.

In a uniform-random network, each peer's in-degree, denoted by *ind*, follows a *Binomial distribution* that $\Pr[ind = k] = C^k_{n-1}p^k(1-p)^{n-1-k}$ where $p = d/(n-1)$. In our simulation that $n = 1000, d = 30$, the variance of *ind* is $\mathrm{VAR}(ind) = (n-1)p(1-p) \approx 30$. Compare with it, outputs $(*, pull, *, head)$, except for $(head, push, push, head)$, have obviously higher variances. Such highly skew networks are undesirable in the case of uniform degree setting.

A network is expected to be dynamic so that it can adapt to the dynamic P2P environment. That is, in each peer's view, other peers shall have a uniform probability to appear. A new notion called *sight* is defined as the number of peers which have at least once appeared in a peer's view. After we start the simulation from a uniform-random network, in most outputs, peers' average sight exceed 200 within 100 time cycles and keep growing. However, in outputs $(head, *, *, *)$, peers' average sight is much smaller and almost stop to grow within 100 time cycles. Such outputs result in severe clustering thus each peer can access only a small part of all peers.

[3]Notice external random factors, e.g. a random executing order, may make outputs more close to a random network than the framework really can do.

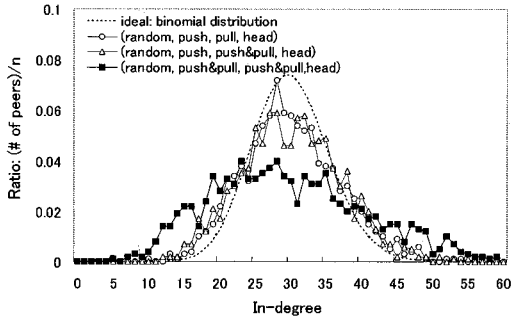

Figure 1: In-degree distribution.

It is possible that a network become highly skew because some unexpected events such as network accident. In this case we expect a network to recover by itself. Such a feature is called self-organization. We start the simulation from a star network which is weakly-connected and has a large in-degree variance . The variance of most outputs, which passed uniform and dynamic tests, can recover, (to the normal in-degree variance), in a short time. However, during the simulation period, the $(*, *, push, *)$ can not even return to strongly-connected while others do within 10 time cycles.

Other tests, such as connectedness, diameter and average path length have also been showed in Table 3.1. As a result, 8 outputs, *(random/tail, push/push&pull,pull/push&pull,head)*, passed all tests. Two outputs *(random/tail, push&pull, push&pull,head)* can also be outputted included in M. Jelasity's framework. Fig.1 shows that some new outputs' in-degree distributions are more close to a uniform-random network than them.

3.2 Degree control

In this simulation, peers are averagely divided into two groups. One group's initial hop count is set to 0 and another is from -1 to -6. We compare the average in-degree of the two groups. The ideal ratio is 2^{-k}, where k is the nonzero initial hop count. According to the degree control principle, if some peers have too low initial hop counts, their

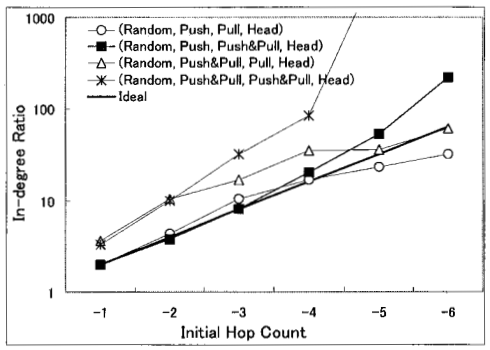

Figure 2: In-degree ration results (random target).

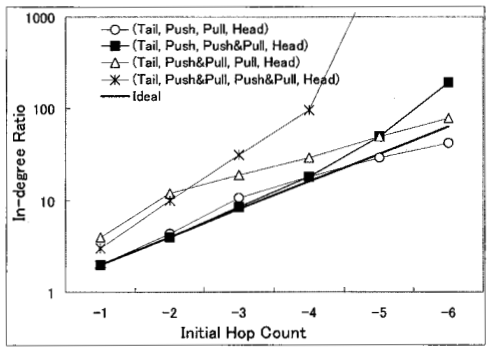

Figure 3: In-degree ration results (tail target).

links will fullfill all peers' views so that normal peers' seeds, that of 0 initial hop counts, can not be inserted into our peers' view. In this case, the degree ratio will higher than we expected.

The results of the remained 8 outputs are show in fig.2 and 3. The curve 'ideal' is the expected ratio. Obviously, only the outputs *(random, push, push&pull, head)* and *(tail, push,push&pull, head)* can correctly control the peers' in-degree as we expected.

4 Concluding remarks

In this paper we proposed a distributed framework to construct degree-controllable P2P overlays. As a result, two outputs of the framework can pass all our tests including degree distribution, dynamic, self-organization and degree control. Although the framework need to be further analyzed, the result shows the possibility of achieving the objective network by only local link exchanges.

Future works:

In this paper, the framework only allow peers have in-degree ratios of powers of 2. It is not difficult to achieve arbitrary ratios. Two approaches are under developing. One is to expand the hop count mechanism to handle non-integer hop counts. Another is to use different execute interval of seed planting process and view merging process.

We also going to test the framework in a dynamic environment that peers join and leave continually. Because the framework can automatically deleted some old links, fault links that point to left peers can not remain in the system for a long time. This feature is so-called self-healing which is a important feature of robust networks. The study of the trade-off between links' fault rate and maintenance cost of the framework is an interesting challenge.

Acknowledgement

This work is supported in part by MEXT: Global COE (Centers of Excellence) Program, JSPS: Grant-in-Aid for Scientific Research ((B)19300017, (B)17300020), JSPS: Grand-in-Aid for Young Scientists ((B)18700059), MEXT: Grant-in-Aid for Scientific Research on Priority Areas (16092215), MIC: Strategic Information and Communications R&D Promotion Programme (SCOPE), Kayamori Foundation of Informational Science Advancement, and the Nakajima Foundation.

参考文献

[1] The Gnutella protocol specification v0.4.

[2] Peter Mahlmann and Christian Schindelhauer. Peer-to-peer networks based on random transformations of connected regular undirected graphs. In *SPAA05*.

[3] `http://gnutella.wego.com`.

[4] KaZaA website: `http://www.kazaa.com`.

[5] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Middleware '04*.