Les Cahiers du GERAD ISSN: 0711–2440

Detecting Good Connections in an Unstructured Peer-to-Peer Network: The Case of Gnutella

Babin, G., Larocque, D.,
A. Dine

G–2006–18

March 2006

Detecting Good Connections in an Unstructured Peer-to-Peer Network: The Case of Gnutella

Gilbert Babin

Service de l'enseignement des technologies de l'information
HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada, H3T 2A7
gilbert.babin@hec.ca

Denis Larocque*, Abdessamad Dine

Service de l'enseignement des méthodes quantitatives de gestion
HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada, H3T 2A7
{denis.larocque;abdessamad.dine}@hec.ca

** and GERAD*

March 2006

Les Cahiers du GERAD

G–2006–18

Abstract

Past studies have shown the unstability of unstructured peer-to-peer networks, in particular Gnutella. Because of this unstability, queries in these networks are inefficient. Furthermore, to keep the integrality of the network, the protocoles used require huge bandwidth usage. One approach proposed by Gnutella conceptors is to create two classes of servents: leaf nodes and ultrapeers. The core of the network is preserved by the ultrapeers, while the leaf nodes remain in the periphery. In this paper, we investigate the possibility to use a different, yet complementary approach, where a node could only keep connections that it deems "good". We show that it is possible to rapidly determine whether a connection is good or not, based on statistics collected on the Gnutella network. The detection of good connections is based on two criteria: the duration of connection establishment time and the number of messages received through the connection.

Résumé

Des études récentes ont démontré l'instabilité des réseaux pair-à-pair non-structurés et en particulier de Gnutella. Par conséquent, les requêtes dans ces réseaux sont inefficaces. De plus, pour conserver l'intégralité du réseau, le protocole utilise beaucoup de bande passante. Une approche proposée par les concepteurs de Gnutella a été de créer deux classes de servents : les nœuds feuilles et les ultrapeers. Le cœur du réseau est préservé par les ultrapeers tandis que les nœuds feuilles demeurent en périphérie. Dans cet article, nous étudions la possibilité d'utiliser une approche différente, mais complémentaire, par laquelle un nœud pourrait ne conserver que les connexions qu'il considère "bonnes". Nous montrons qu'il est possible de déterminer rapidement si une connexion est bonne ou pas en se basant sur des statistiques collectées sur le réseau Gnutella. La détection des bonnes connexions est basée sur deux critères : le temps de création de la connexion et le nombre de messages qui transitent par la connexion.

1 Introduction

Peer-to-peer (P2P) networks have been profusely studied in the recent past. The focus of many of the research projects on this topic has been on ways to improve performance of the network. Performance here is mostly measured by the efficiency of search operations on the network and by bandwidth usage to preserve the network structure. This lead to a dichotomy of P2P systems, namely, unstructured and structured P2P networks.

In unstructured networks (e.g., Napster, Gnutella [1], FreeNet [2]), performances are not very good : bandwidth usage is usually atrocious [3, 4, 5, 6, 7] and it is not guaranteed that one finds what he is looking for. However, these networks require no authentication nor collaboration from the participating peer which may be seen as an advantage in many contexts.

Structured networks (e.g., P-grid [8], BALLS [9, 10], PAST [11, 12], CAN and Expressways [13, 14], and others [15, 16, 17, 18]) were designed to offer efficient search mechanisms (often $O(\log n)$) while using as little bandwidth as possible. To achieve these performances, peers must collaborate and therefore, some kind of authentification is required.

Clearly, there are circumstances that call for less structure and more freedom on the part of the participating nodes. In these cases, unstructured P2P, and more specifically Gnutella protocol-based networks, are appropriate, but their performances are not appealing. The question therefore is how to improve unstructured P2P networks' performances.

We conducted a study of the Gnutella network in 2001 [3]. The study showed the volatility of the Gnutella network. In particular, we observed that the network is quite unstable, with a median connection duration of 0.17 seconds. Designers of Gnutella client software (such as LimeWire [19]) have modified the protocol to limit the impact of such volatility. The idea is to promote peers that stay connected for a long period to the status of ultrapeers. The core of the network is preserved by the ultrapeers while other peers (leaf nodes) remain at the periphery.

This paper looks at the problem from a different perspective. We try to assess whether it is possible to prune bad connections (i.e., connections that will not last) early so as to favor good connections, and consequently provide a more stable network.

We focus on the Gnutella network as it is often cited as a typical unstructured P2P network without central control. Furthermore, the protocol has been the subject of many studies, articles, etc., as demonstrated by the 47,300 hits on the topic "Gnutella protocol" in Google. Finally, it was fairly easy to obtain a robust version of the software that we instrumented for the purpose of this study. In the next section, we describe the research problem. Section 3 describe the experimental approach used to collect data and the analysis and results are presented in Section 4. We then conclude the paper with a discussion on the results obtained in Section 5.

2 Good Connections in the Gnutella Protocol

The Gnutella protocol allows peers (a.k.a. servents for *serv*er and cli*ent*) to exchange files without any central server acting as a catalogue or a dispatcher. A peer opens a user-specified number of TCP connections with other peers. All communications in the network occur through these connections. Hence, the Gnutella network is laid over the Internet.

Nodes constantly make sure that the current number of active connections correspond to the maximum number of permitted connections. This is how the peers stay connected to each other. As stated above, however, the median duration of a connection is quite small. Consequently, a peer spends a large portion of its active life reconnecting to the network.

The Gnutella protocol is very simple in that queries (QUERY message type) and keep-alive (PING message type) messages are flooded in the network through the TCP connections already established. The flooding process is controlled via a "time to live" (TTL) field which is decremented at every intermediate peer. Answers to query (REPLY message type) and keep-alive (PONG message type) messages are routed back to the originating peer; i.e., the answer follows the reverse sequence of TCP connections used for the query/keep-alive message to reach its destination.

In the newest version of the protocol, servents may be leaf nodes or ultrapeers. Leaf nodes may only connect to ultrapeers. The number of such connections is user-determined. Ultrapeers connect with other ultrapeers and allow leaf nodes to establish connection with them. Again, the number of ultrapeer and leaf node connections allowed is specified by the user.

In this version, the ultrapeers form the core of the network and therefore must provide the most stable and reliable network possible. Clearly, establishing long lasting connections should provide better stability and more reliability. Indeed, as replies are routed back through the same TCP connections, should a connection be broken, all replies routed through that connection are lost. It follows that the stability of the network and the efficiency of searches is highly dependent on the duration of connections. Consider PING messages, which are sent to keep track of the network composition. Should the average duration of connections be increased, the frequency at which PING are sent could be reduced. Furthermore, the number of lost replies (replies that cannot be routed back to their origin because of broken connections on the return path) will be reduced if connections remain open longer. Therefore, a *good connection* is a connection that remains open as long as possible. As this definition of good refers to some dichotomy, we simply fix a threshold to classify connections as good ($duration \geq threshold$) or not ($duration < threshold$).

The problem is that we cannot tag a connection as good until it is broken. Or is it the case? This is exactly the question that we address in this paper, that is, whether or not there are any criteria that could be used to rapidly determine if a connection will be good or not. If such criteria exist, are they dependent on the user specified parameters?

The next section describes the experimental approach used to collect data on the Gnutella protocol, while Section 4 examines strategies to classify a connection as good or not based on the statistics collected.

3 Data collection

We wanted to collect data about Gnutella in the least intrusive manner. Consequently, we used a publicly available servent (namely LimeWire [19]) and instrumented it. We did not change the operations of the servent. We simply added data collection routines that enabled us to save detailed statistics about the servent, its network usage, all connections established, all messages sent/received. As such, our servent was seen as a regular LimeWire servant on the network and behaved as such.

The modified servent was able to collect the following data each time a servent is executed :

- Statistics on all connections established : duration, termination code.
- Statistics on all messages received/sent : type, size, hopcount, TTL, date received.
- A list of all queries performed on the network while the servent was active[1].
- Statistics on the servent : bandwidth, horizon (number of reachable servents, files), connections attempts, received messages, routed messages, sent messages.

In order to collect a significant amount of data, we ran our modified servent a number of time and followed the experimental protocol proposed in [3]. The experiments are performed in runs, once per hour for 24 hours. In each run, two servents are started in parallel, both servents being in the same mode either in leaf node mode or ultrapeer mode. One servent, the *benchmark peer*, uses base parameters (the same for all runs) while the other servent, the *test peer*, uses different user-specified parameters (see Tables 1 and 2)[2]. Each run lasts for 45 minutes. All experiments were performed in the period from July 21, 2003 to August 25, 2003. In this paper, we limit our analysis to ultrapeers.

4 Data analysis and results

This section presents the results from our experiments. For the remainder of the paper, a connection is considered good if its duration exceeds 30 seconds. The value of the threshold was fixed arbitrarily such that about 35% of connections would be classified as good and that the duration of connections was reasonably long.

[1] For obvious reasons, we did not share any content on the network. We only observed queries in transit through our servent.

[2] Data from run 4 are unavailable and therefore this combination of parameters will not be considered in this study.

Table 1: User-specified parameter in leaf node mode

Run	*Connections to ultrapeers*
1	1
2	2
3	3
4*	4
5	5
6	7
7	10
8	15

* *benchmark peer* values

Table 2: User-specified parameters in ultrapeer mode

Run	*Connections to ultrapeers*	*Connections from leaf nodes*
1	10	30
2	22	30
3	25	30
4	28	30
5	30	30
6*	32	30
7	34	30
8	36	30
9	39	30
10	42	30
11	54	30
12	32	60

* *benchmark peer* values

The total number of connections in each run varied between 15274 and 21107. Figure 1 provides the proportions of good connections in each run for both the test and benchmark peers. We clearly see a run effect (the spikes in run 2 and 11) in the sense that the proportions are not stable across runs. This justifies the use of a benchmark peer as a control. We compared the proportions of good connections of the benchmark and test peers on a run by run basis. At the 1% level, the proportions are significantly different for run 1, 11 and 12 only.

Our goal is to identify rules that can detect good connections as fast as possible. With a view towards real applications, these rules should be simple, fast to compute and easy to implement for practical use.

Exploratory analysis using the aggregated benchmark peer data indicated that a bad connection possesses in general two simple characteristics:

1. A longer time to establish the connection (creation time).
2. No activity through the connection early on, that is, no messages of any type are received in the first few moments after the establishment of the connection.

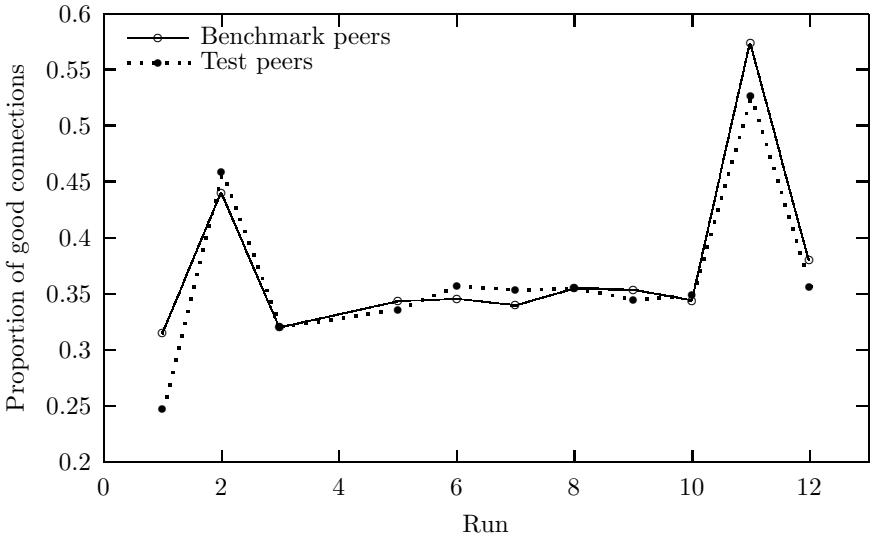

Figure 1: Proportion of good connections vs. runs for test and benchmark peers

Tables 3 and 4 clearly demonstrate those facts. Note that in the next 3 tables, the first number in each cell is the raw frequency while the second one is the column percentage. Table 3 presents the cross-tabulation of the duration and a dichotomized version of the variable creation time. This variable, named *creation*, takes the value 0 if the creation time is smaller than 1 millisecond (ms) and the value 1 if it is greater than or equal to 1 ms. Note that the ms was the unit of measure so it was not possible to go beyond that. On one hand, we see that 83.4% of the connections with a creation time less than 1 ms turned out to be good connections. On the other hand, 67.3% of the connections with a creation time greater than or equal to 1 ms turned out to be bad connections. It is important to note that in order to make it comparable with Table 4, the classification of each connection was done 1 second after its establishment. Consequently, the (bad) connections that did not last 1 second are not included in Table 3. For Table 4, a binary variable named *activity* was created according to the fact that no message at all transited through the connection during the first second after its establishment (0) or that at least one message of any type transited (1). We see 81.5% of the connections that had any activity after 1 second turned out to be good connections compared to only 55.3% for the connections that did not have any activity. It is clear from these two tables that both variables discriminate between good and bad connections but that *creation* has more discriminating power.

Table 5 shows that both variables could also be used together. It presents the same cross-tabulation as Table 4 but only for the connections for which the creation time is greater than or equal to 1 ms (*creation*=1). We see that 70.1% of the connections for which *creation*=1 and for which no messages at all transited during the first second turned

Table 3: Cross-tabulation of duration time and creation time

	creation (creation time)	
	0 ($\leq$ 1 ms)	1 ($>$ 1 ms)
Good connection	27713	8635
	83.4	32.7
Bad connection	5500	17770
	16.6	67.3

Table 4: Cross-tabulation of duration time and activity (after 1 second)

	activity (# of messages)	
	0 ($=0$)	1 (>0)
Good connection	25829	10519
	55.3	81.5
Bad connection	20878	2392
	44.7	18.5

Table 5: Cross-tabulation of duration time and activity (after 1 second) for the connection with creation time $\geq$ 1 ms (*creation*=1)

	activity (# of messages)	
	0 ($=0$)	1 (>0)
Good connection	6998	1637
	29.9	54.1
Bad connection	16382	1388
	70.1	45.9

out to be bad connections. Moreover, 54.1% of the connections for which *creation*=1 but for which at least one message transited during the first second turned out to be good connections. This combination of criterion could be used to recover some good connections and increase the sensitivity of the rule. Even if the above tables show the results when we classify a connection after 1 second, it is clear that different time windows can be used and this is explored below.

Based on the facts above, two simple rules were built. For both rules, the classification of a connection occurs w (≥ 0) seconds after it is established so both rules are in fact a family of rules that depend on the window length w.

- Rule A_w: Automatically classify a connection as bad if it terminated before w seconds. If the connection is still alive, classify it as good if its creation time is smaller than 1 ms and as bad if it is greater than or equal to 1 ms.

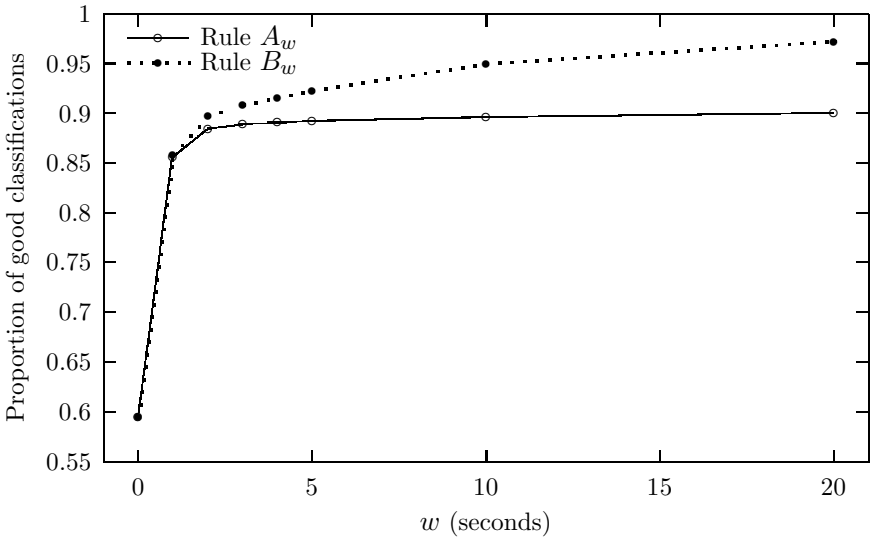

Figure 2: Proportion of good classifications vs. w for rules A_w and B_w

- Rule B_w: Automatically classify a connection as bad if it terminated before w seconds. If the connection is still alive, classify it as bad if its creation time is greater than or equal to 1 ms and no messages at all have transited through it so far. Otherwise, the connection is classified as good.

It should be clear that applying Rule A_w after 1 second will not classify the connections in the same way as applying it after 2 second since some connections will be lost in the interval [1,2] and those are automatically correctly classified as bad connections. Moreover, Rule A_0 is identical to rule B_0.

Again using the aggregated benchmark peer data, Figure 2 presents the proportions of good classifications for both rules as a function of w. We see that waiting a little before making a decision greatly improves the performance of the rules. The good classification rate is under 60% with $w = 0$ and jumps above 85% for both rules when $w = 1$ (85.6% for Rule A_1 and 85.8% for Rule B_1). At $w = 2$ seconds, we have good classification rates of 88.4% and 89.7% for rules A_2 and B_2 respectively. After that, the relative gain of waiting more before making a decision about a connection is smaller.

Since we are really interested in detecting good connections, we will examine the sensitivity of the rules which is defined in our case as the proportion of good connections that are well classified (it measures how well the rule detects good connections). Figure 3 presents the sensitivity of the rules as a function of w. The sensitivity of Rule A_w is constant because the set of good connections is always the same for every window length. The sensitivity of both rules is 76.2% at w=0. This means that 76.2% of the good connections

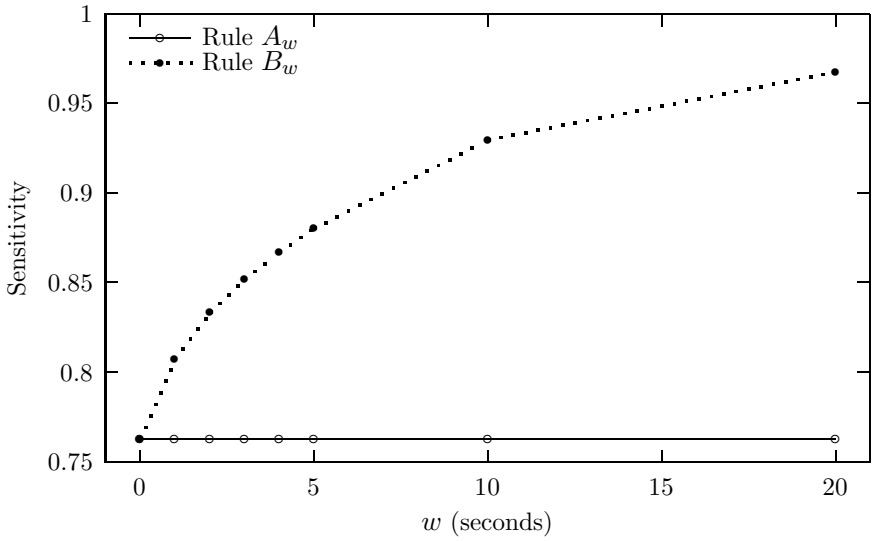

Figure 3: Sensitivity of classification vs. w for rules A_w and B_w

are well classified. The sensitivity of Rule B_w increases to 80.8% and 83.4% at times w=1 and 2 respectively. Since the goal is to identify the good connections as fast as possible, these results indicate that making a decision after w=1 or 2 seconds using Rule B_w could be appropriate. In practice, the definitive choice of w could depend on other considerations.

The performance reported above were obtained with the benchmark data. The next step is to investigate how those performances are dependent on the user-specified parameters. We selected w=1 as window length and compared rule B_w across the different runs. Recall that in each run, data are available for a benchmark peer that always uses the same user-specified parameters and for a test peer that uses different parameters. The rule B_1 was applied separately for each peer and for each of the 11 runs. Figure 4 presents the 22 proportions of good classification obtained. For reference, the proportion of good classification (0.858) obtained with the aggregated benchmark data is also shown on the graph. Apart from run 11, the proportions seem pretty stable. We compared the proportions of good classification of the benchmark and test peers on a run by run basis. At the 1% level, the proportions are significantly different for run 2 and 11 only. This indicates that the performance of the rule does not seem to depend very much on the user-specified parameters. The sensitivities are depicted in Figure 5 along with the one obtained with the aggregated benchmark data (0.808). Once again, the sensitivity seems quite stable with the exception of run 11.

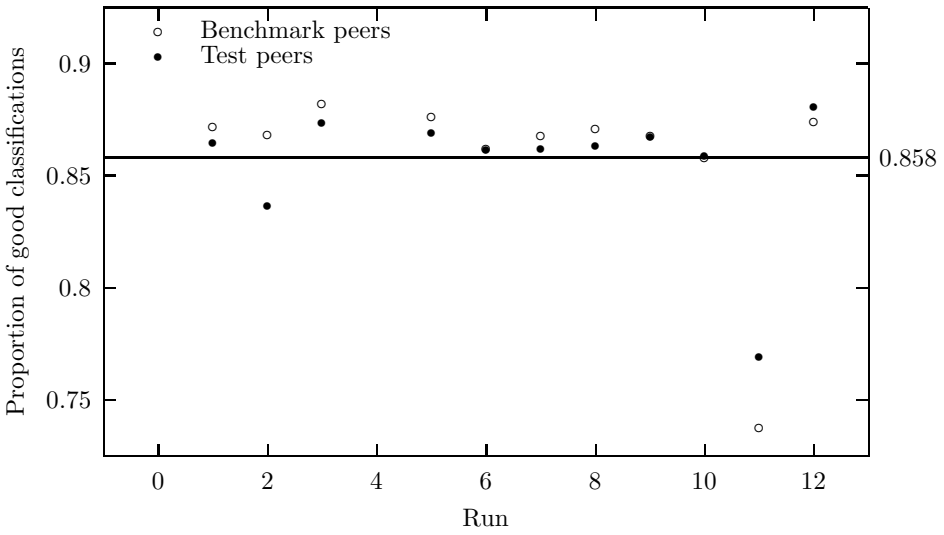

Figure 4: Proportion of good classifications vs. run using rule B_1

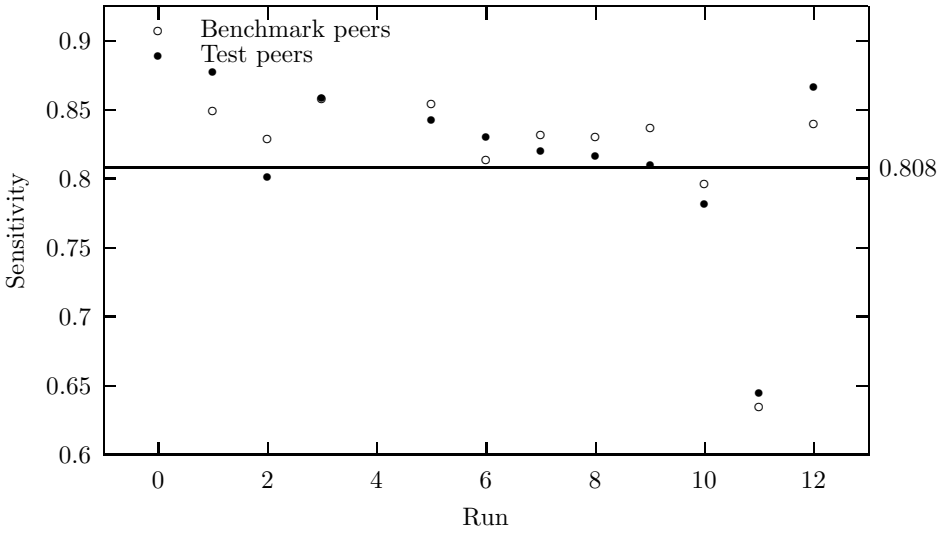

Figure 5: Sensitivity of classification vs. run using rule B_1

5 Conclusions

The analysis performed clearly shows that a servent could apply either rules B_1 or B_2 on all connections and only keep those classified as good, which could therefore increase the stability of the network. Consequently, the frequency at which PING messages are sent could be reduced, hence having a direct impact on bandwidth usage. Furthermore, the increased stability will reduce the number of wrongly aborted replies, as more replies will reach the originator of the request.

This conclusion is further supported by results obtained in a previous study [20], where the authors demonstrated that for the same data, the message traffic of the Gnutella protocol is self-similar[3]. Hence, a short time interval should provide sufficient information to predict the behavior over a long period. This is especially true when considering the *activity* variable, as no activity early on leads to little or no activity later.

The implementation of rule B_w is fairly simple. Every time a connection is attempted, we can record the creation time. Then, once the connection is established, we launch a timer thread that will expire after w seconds. If no messages are received through the connection after that delay and the creation time was longer or equal to 1 ms, the connection is closed.

We observed that a large number of connection attempts have a very long connection time which roughly corresponds to the TCP layer time out. Consequently, it may be argued that connections that take a long time to establish are on less stable routes, and therefore are more likely to produce time-outs. It also follows that any traffic on these connections will also be difficult to maintain, once established, as it will follow the same route to reach its destination. Hence, by dropping those connections with a long connection time and no traffic early on, we basically prune connections that will always show difficulties.

In this paper, we presented a simple approach to detect good connections in the Gnutella network, based on empirical evidence. We claimed that by only keeping such good connections, network stability and search efficiency would be improved. We also speculated that long connection times lead to bad connections, as long connection times are symptomatic of bad routes. We did not, however, provide any empirical results that would directly support these claims and speculations. Clearly, more work is required in that respect and should be addressed in future studies.

[3]Self-similar traffic is defined as traffic pattern that is invariant against changes in scale or size [21, 22].

References

[1] "Limewire llc," LimeWire LLC, 2002, http://www.limewire.com/.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "freenet: a distributed anonymous information storage and retrieval system," in *ICSI Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, 2000. [Online]. Available: citeseer.nj.nec.com/350305.html

[3] J. Vaucher, G. Babin, P. Kropf, and T. Jouve, "Experimenting with gnutella communities," in *Conference on Distributed Communities on the Web (DCW 2002)*, ser. Lecture Notes in Computer Science, no. 2468. Sydney, Australia: Springer Verlag, Apr. 2002, pp. 85–99. [Online]. Available: pub/Vauc02a.pdf

[4] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *1st IEEE International Conference on Peer-to-peer Computing (P2P2001)*, Linkoping, Sweden, Aug. 2001.

[5] J. Ritter, "Why gnutella can't scale. no, really," 2001, http://www.darkridge.com/~jpr5/doc/gnutella.html.

[6] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy, "An analysis of internet content delivery systems," in *OSDI'02*, 2002.

[7] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *ACM SOSP'03*, Oct. 2003.

[8] K. Aberer, "P-grid: A self-organizing access structure for p2p information systems," in *CoopIS 01*, 2001.

[9] V. D. Le, G. Babin, and P. Kropf, "A structured peer-to-peer system with integrated index and storage load balancing," in *Innovative Internet Community Systems (I2CS) 2005*, Paris, France, June 2005.

[10] ——, "Balls simulator: Evaluator of a structured peer-to-peer system with integrated load balancing," in *Research, Innovation and Vision for the Future (RIVF'06)*, Ho Chi Minh City, Vietnam, Feb. 2006.

[11] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *IFIP/ACM Middleware'01*, Nov. 2001.

[12] ——, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *ACM SOSP'01*, Oct. 2001.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalabale content-addressable network," in *ACM SIGCOMM'01*, Aug. 2001, pp. 161–172.

[14] Z. Zhang, S.-M. Shi, and J. Zhu, "Self-balanced p2p expressways: When marxism meets confucian," Microsoft Research, Tech. Rep. MSR-TR-2002-72, 2002.

[15] M. F. Kaashoek and D. R. Karger, "Koorde: A simple degree-optimal distributed hash table," in *IPTPS'03*, Feb. 2003.

[16] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *ACM PODC'02*, July 2002, pp. 183–192.

[17] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in structured p2p systems," in *IPTPS'03*, Feb. 2003.

[18] I. Stoica, R. Moris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM'01*, Aug. 2001, pp. 149–160.

[19] C. Rohrs, "Limewire design," LimeWire LLC, 2001, http://www.limewire.org/techdocs/design.html.

[20] D. Tutsch, G. Babin, and P. Kropf, "Application layer traffic analysis of a peer-to-peer system," in *The Montreal Conference on e-Technologies*, Montreal, Canada, May 2006.

[21] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–14, Feb. 1994.

[22] W. Stallings, *High-Speed Networks*. New Jersey: Prentice Hall, 1998.