# Gotta Catch'em All! Improving P2P Network Crawling Strategies

Alexander Mühle
*Hasso-Plattner-Institute*
*University Potsdam*
Potsdam, Germany
alexander.muehle@hpi.de

Andreas Grüner
*Hasso-Plattner-Institute*
*University Potsdam*
Potsdam, Germany
andreas.gruener@hpi.de

Christoph Meinel
*Hasso-Plattner-Institute*
*University Potsdam*
Potsdam, Germany
christoph.meinel@hpi.de

***Abstract*—Network crawling has been utilised to analyse peer-to-peer systems by academics and industry alike. However, accurately capturing snapshots is highly dependant on the crawlers' speed as the network can be described as a moving target. In this paper, we present improvements based on the example of a newly developed Bitcoin crawler that can be utilised to reduce resource usage/requirements of crawlers and therefore speed up capturing network snapshots. To evaluate the new strategies, we compare our solution, in terms of increased scan rate and increased hit rate during crawling, to a popular open-source Bitcoin monitor. Blocking time is reduced on average to 1.52s, resulting in 94.7% higher scan rates, while time needed to capture a network snapshot is reduced on average by 9% due to increased hit rates during network crawling. While we show our improvements at the example of a new Bitcoin crawler, proven concepts can be transferred to other P2P networks as well.**

***Index Terms*—peer-to-peer systems, blockchain, network crawling, internet measurement**

## I. Introduction

Peer-to-Peer applications have seen a resurgence of popularity in recent years, often in the context of cryptocurrencies such as Bitcoin. Due to this trend, the research community, as well as commercial interests, have undertaken numerous projects to measure these kinds of networks.

Crawling networks is an inherently progressive process, yet the goal is to capture a snapshot of the current network and its participants. Ideally, a snapshot would include all current online nodes exactly. This process, however, is impacted by several factors.

Typically crawlers join a given network and participate in the node discovery protocol, recursively contacting all other discovered peers. Churn, the effect of peers constantly leaving and joining such open networks, is a key contributor to inaccuracies of these measurements. The network can be described as a moving target, during the duration of the crawl, it is already changing, and when finished will not exactly represent the real state of the network. Due to this continuous fluctuation, the speed of a crawl has a direct and significant impact on the accuracy. Especially in cryptocurrency networks such as Bitcoin the responses to address requests are random and only a portion of the known peers of a contacted peer are returned. This also means repeated crawling will increase the likelihood of a more complete view of the network.

Additionally, in most peer-to-peer networks, there are no explicit exits from the network, peers simply disconnect from their respective neighbours; this change, however, is not widely propagated. This leads to addresses of such nodes often lingering in the network for a considerable time. Generally the majority of peers will be unreachable by crawlers, either due to exiting the system or being unreachable in general. As a consequence, during crawling rounds, these addresses waste resources of potential crawlers through connection attempts that will timeout and are therefore costly.

In this paper, we contribute to the quest to an accurate network snapshot of large open peer to peer systems such as Bitcoin by improving the strategies used for such crawling. For this purpose we implement scale-out through a manager/worker architecture. Through network measurements we find optimal timeouts and in order to reduce predictably unsuccessful connections altogether we take into account previous offline time of nodes as well as perform Bogon filtering.

We will evaluate these in terms of decreased timeouts and for the trade-off between increased hit rate and discovered peers.

## II. Related Work

Since the first large wave of peer-to-peer applications, the exploration of the associated network has been of interest to researchers. In the following, we will hence discuss both efforts from the general peer-to-peer context, often with the use-case of file-sharing, and more recent efforts looking at cryptocurrency networks.

### A. Peer-to-Peer Crawler

Gnutella was one of the first widely analysed peer-to-peer networks. Ripeanu et al. underwent a study to map the Gnutella network [1]. The paper tackles two main questions in regards to the then relatively new approach of peer-to-peer networks. Fault tolerance and robustness of the network as well as exploring the mismatch between virtual overlay network and phsical internet infrastructure. For this purpose, they developed a crawler that uses the membership protocol of Gnutella to collect peer information. They utilised an initial list of nodes to contact, which then progressively gets extended by neighbour information of newly contacted peers. In order

to speed up the crawling process, Ripeanu et al. used a manager/worker architecture where the server was responsible for assigning IPs to clients to contact.

In a very similar fashion, Stutzbach et al. [2] developed *cruiser* which they aimed to be an improved network crawler in order to capture more accurate snapshots of the Gnutella Network. They saw five key areas where this performance increase could be achieved. Inherent to the Gnutella protocol were two areas: handshaking and the two-tier structure of the Gnutella network. Just as Ripeanu before them, they also deployed *cruiser* as a distributed system with manager/worker architecture. Finally, they recognised that appropriate timeouts were essential for a performant crawler as non-responsive peers were a significant percentage (30%-38%) of overall contacts in the Gnutella network.

In addition to the already described approaches, Deschenes et al. [3] also had a running listener in their crawler, which was open to connections. During the membership protocol, the crawler would announce the IP/port of the listener and potentially solicit new connections from unknown peers.

Crawling in the above-described way is not limited to file-sharing networks like Gnutella or eDonkey [4] but has also been employed in VoIP programs such as Skype [5] or IPTV systems [6].

While the previously described crawlers all used a crawling strategy of only going through a queue of available peers, Saroiu et al. [7] followed a different strategy. Rather than crawling through the complete list of available peers, they limited the crawl time per snapshot to two minutes and then restarted the crawling process with the updated list of available peers. This, however, means that they only gather 25%-50% of the total population of peers in the system.

### B. Bitcoin Crawler

In more recent times the most prominent peer-to-peer network that has been analysed with the use of crawlers has been Bitcoin. The measurement studies had different focuses, from the basic makeup of the network [8], [9] to analysis of information propagation [10], [11] and inference of topology [12], [13] while others focused on deanonymisation of participants [14] [15].

Similar to previous peer-to-peer crawlers, these crawlers utilised progressive crawling through available peers. For this purpose, either existing Bitcoin clients were extended for logging capabilities or dedicated crawler software was developed [15], [16]. However, to our knowledge, most utilise a single instance approach rather than a distributed manager/worker architecture, and like most previous efforts they stick to a simple crawling strategy contacting all available nodes in each snapshot run.

Some monitors are publicly available such as the KIT DSN Bitcoin monitor [1] and Bitcoinstats.com, yet to our knowledge of the popular currently running monitors only Bitnodes publishes their code [2]

[1] dsn.tm.kit.edu/bitcoin

[2] github.com/ayeowch/bitnodes

## III. Design Considerations

### A. Architecture

The Bitcoin crawler developed for this project was written as a multi-processing Python3 application. It utilises non-blocking, asynchronous I/O in order to maximise the number of concurrent connection attempts. In order to further increase crawling capacity for a single snapshot, we chose to implement a manager/worker architecture. The crawler, therefore, consists of a central coordinator instance and a variable number of crawling instances. The initial resolution of seed nodes, which are the entry point into the peer-to-peer network, is done at each individual crawling instance. However, all results of address requests are sent to the central coordinator, which then, in turn, gives out new tasks for the crawlers, so a single snapshot is created. Each crawling instance has a local cache of already discovered nodes. The cache minimises network traffic as these nodes do not have to communicated to the coordinator again. Only new nodes or nodes which had a change in status (online/offline) are sent to the coordinator. The workers themselves are made up of two different subsystems - the active participant in the neighbour discovery protocol and the passive listener. The active connector makes address requests to other peers while at the same time announcing the IP/port of the passive listener. Through this, the addresses of our crawlers are propagated throughout the network. These announcements can lead to unsolicited connection attempts by potentially previously unknown, especially new, peers in the network.

### B. Bogon Filtering

An improvement to the hit rate of our crawler is to employ Bogon filtering. The filtering of private or reserved subnets as well as IP space that has not been assigned to an ISP yet is a common practice for firewalls. These are all addresses that should not be reachable on the internet. We utilise IP lists provided by Team Cymru [3]. In regular operation, these should rarely be propagated by Bitcoin peers. The Bitcoin reference implementation categorises peers in two categories, *good* and *terrible*. When an address request gets answered, a random selection of addresses is selected, but *terrible* nodes are skipped. Four different checks characterise these *terrible* nodes: a timestamp more than 10 minutes in the future, not seen in 30 days, never responded after three attempts or seven separate failures during a week. Nonetheless, there can be instances where unreachable node addresses (i.e Bogons) get propagated, such as very new addresses (addresses known less than a minute are exempt from the *terrible* rating) or a node might be reachable in a private network and therefore not receive a *terrible* rating yet be unreachable for our crawler from the internet.

### C. Status Checking

Although there might be some Bogons in the Bitcoin network peer databases, it is much more likely that we encounter

[3] team-cymru.org

valid but unreachable IPs. Network address translation often used for consumer internet connections, firewalls or peers that simply left the network and do not run the peer-to-peer application in question anymore, are typical examples. However, these peers might be only temporarily unreachable, which is why we will have to periodically check their status in subsequent snapshots in order to keep an accurate picture of the network. In our experiments, of the reachable peers most were reachable during our first connection attempt. However, a third of peers that we found to be online at some point only came online after we started our crawl. Interestingly the number of IPs that re-entered the network after an offline time was relatively low with 3%. We suspect that most residential setups which are more likely to enter and exit the network regularly are already not reachable due to NATs. In Figure 1 the offline time of peers that either re-entered or came online during our crawl can be observed.

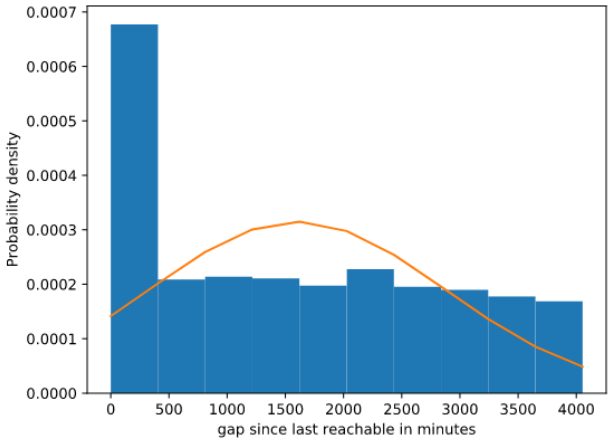


Fig. 1. Time until successful connection establishment

In order to further improve the performance of our crawler, we will use a kind of ”back-off” on our status checks. For this purpose, the central coordinator keeps track of the offline time for each participant. The longer a peer has been offline, the less likely it is that it will come back and be reachable. Therefore the coordinator will also be less likely to assign an IP to a crawler if said IP has been offline for a longer time, minimising likely unsuccessful connection attempts. This is implemented by comparing the duration since the *last_checked* date with the *offline_time* at the last check If the duration since the last check exceeds the offline time, multiplied with a *back-off factor*, at the last check, then a new check is performed.

### D. Optimal Timeout

As mentioned before there is a large portion of the network that will never be reachable for our crawlers. This includes already departed peers as well as peers that are unreachable behind a NAT or similar middleboxes. In these cases, the typical behaviour would be to wait for the *tcp_syn_retries* of the kernel to timeout and detect a failed connection establishment attempt. With the default settings of most Linux distributions, an initial connection timeout takes around 45 seconds. In order to reduce the required timeout, we measure the behaviour of alive peers to find the optimal trade-off between speed and completeness of the crawl. In Figure 2 (a) this behaviour of alive peers can be observed. The time in seconds before a connection is established is shown as a boxplot. The median time is at 73ms while the average is around 110ms with a max of 380ms. In order to capture the majority of attempts but not sacrificing performance in order to capture all outliers, we chose the timeout for the initial connection establishment as trade-off at 1 second.

In order to improve the performance not only for unreachable peers but reachable peers as well, in addition to the *tcp_syn_retries*, we consider the TCP connection timeout of already established connections. If there is ”silence on the wire”, it could take a standard Linux host around 2 hours to terminate the connection unless otherwise instructed. Finding a fitting timeout, therefore, is quite beneficial. In contrast to the *tcp_syn_retries* timeout, our idle timeout is considerably longer as it is not dominated by latency but rather the processing/queue at the client of the opposite peer. In Figure 2 (b)/(c) our measured duration of request responses can be seen. In (b) only the minimum measurements of each node are considered while (c) is a representation of all measurements. It shows that as a minimum half of all measurements are below 1.51s while the upper half of measurements goes up to 13s. When considering all measurements, however, we observe a considerably higher spread of delays. As these are only outliers and subsequent or previous requests were answered more quickly, we opt to choose the timeout according to the minimum latencies per node not overall. Hence the connection timeout for ongoing connections is set to 13s.

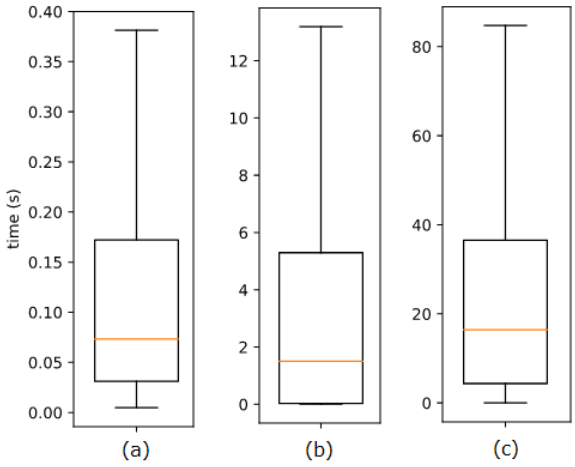


Fig. 2. (a) Time until successful TCP connection establishment (b) Minimum time per node until successful address request response (c) Distribution of all measurements for address request-response

## IV. Data Gathering

We deployed our crawlers over a mixed infrastructure of university and public cloud infrastructure. Amazon Web

Services was used to deploy crawl-workers to three geographically distributed areas, in particular, Ireland, Ohio and Hong Kong to cover three continents. Additionally, the coordinator and another crawler were hosted at the Hasso-Plattner-Institute in Potsdam, Germany.

## V. Evaluation

### A. Increased Scan Rate

In order to compare the performance of our crawler and specifically the impact of our chosen timeout, we took a popular Bitcoin crawler that is open source as baseline [4]. Bitnodes chooses a single timeout for both the establishment as well as the idle connection, which by default is 30 seconds, compared with our 1s initial and 13s idle connection timeout. In Table I we show in which phase of the connection during our experiment run, connections actually timeout.

| Phase | Percentage |
|---|---|
| TCP connection establishment | 95.6% |
| BTC handshake | 2.4% |
| BTC address request | 1.7% |

TABLE I
PHASE OF TIMEOUT IN CONNECTION

It can be observed that the vast majority of peers already fail during the initial TCP connection establishment. This means our improved timeout during this phase gives us a significant performance improvement. Due to the high number of peers behind NATs and firewalls unreachable to our crawler as well as the significant churn in the network, the vast majority (96.2%) of discovered nodes are never reachable and hence timeout. For this majority, our crawler overall (combining all phases and their probability) has an average 1.52s timeout (94.93% lower compared to Bitnodes). Using the same resources (number of co-routines) this leads to a 94.7% faster scan rate.

### B. Increased Hit Rate

One of our tactics to increase crawling speed is to not only have lower timeouts but avoid predictable timeouts altogether. The propagation of Bogons in the Bitcoin network seems to be minimal as during a 24h crawl only 18 Bogons, mainly from private subnets, have been discovered. The increase in hit rate from Bogon filtering is, therefore, negligible. To evaluate the overall effectiveness of our measures, we compare the hit rate as well as discovered new peers of the typical consecutive snapshot strategy with our strategy that takes into account previous offline time. For this purpose, we ran our crawlers with and without the "back-off" strategy for at least 10 consecutive snapshots. The evaluation showed that the back-off strategy indeed increased the hit rate from 3.8% to 4.8% while the amount of discovered new peers stayed consistent with previous efforts. The crawl speed increase due to our back-off strategy was on average 9%.

[4] bitnodes.io

## VI. Conclusion

We developed a new Bitcoin crawler that decreases blocking time through optimised timeouts as well as increased hit rate during consecutive network snapshots. Through our evaluation, we quantified this improvement. Our Bitcoin network crawler increases scan rate by 94.7% compared to other publicly available Bitcoin monitors and increases the hit rate from roughly 3.8% to 4.8% due to taking into account the previous offline time of a node before including it in the crawl list. This new strategy also decreases the creation time of a snapshot by 9%.

## References


[1] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *arXiv preprint cs/0209028*, 2002.

[2] D. Stutzbach and R. Rejaie, "Capturing accurate snapshots of the gnutella network," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*. IEEE, 2006, pp. 1–6.

[3] D. G. Deschenes, S. D. Weber, and B. D. Davison, "Crawling gnutella: Lessons learned," *Technical Report*, 2004.

[4] J. Yang, H. Ma, W. Song, J. Cui, and C. Zhou, "Crawling the edonkey network," in *2006 fifth international conference on grid and cooperative computing workshops*. IEEE, 2006, pp. 133–136.

[5] S. Guha and N. Daswani, "An experimental study of the skype peer-to-peer voip system," Cornell University, Tech. Rep., 2005.

[6] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A measurement study of a large-scale p2p iptv system," *IEEE transactions on multimedia*, vol. 9, no. 8, pp. 1672–1687, 2007.

[7] S. Saroiu, K. P. Gummadi, and S. D. Gribble, "Measuring and analyzing the characteristics of napster and gnutella hosts," *Multimedia systems*, vol. 9, no. 2, pp. 170–184, 2003.

[8] J. A. D. Donet, C. Pérez-Sola, and J. Herrera-Joancomartí, "The bitcoin p2p network," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 87–102.

[9] T. Neudecker, "Characterization of the bitcoin peer-to-peer network (2015-2018)," *Karlsruhe, Tech. Rep.*, p. 1, 2019.

[10] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*. IEEE, 2013, pp. 1–10.

[11] R. Kanda and K. Shudo, "Estimation of data propagation time on the bitcoin network," in *Proceedings of the Asian Internet Engineering Conference*, 2019, pp. 47–52.

[12] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering bitcoin's public topology and influential nodes," *et al*, 2015.

[13] T. Neudecker, P. Andelfinger, and H. Hartenstein, "Timing analysis for inferring the topology of the bitcoin peer-to-peer network," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. IEEE, 2016, pp. 358–367.

[14] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 15–29.

[15] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in bitcoin using p2p network traffic," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 469–485.

[16] D. D. F. Maesa, M. Franceschi, B. Guidi, and L. Ricci, "Bitker: a p2p kernel client for bitcoin," in *2018 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2018, pp. 130–137.