Comparative Analytic Simulation of Peer to Peer Networks

M.S. Gaur, A. Narayanan, R. Murmuria

Malaviya National Institute of Technology
Department of Computer Engineering
JLN Marg, Jaipur, India

msgaur@mnit.ac.in, anant@kix.in, rahul.murmuria@gmail.com

Abstract

Peer to Peer networks have increasingly grown in significance. They have proven to be one of the most effecient methods of transferring large amounts of data across a distributed network, evenly distributing the load. Several protocols have been developed over the last few years, and this paper attempts to provide a brief overview of the various networks that have evolved from these protocols on the basis of analytical simulation. Given the significance of peer to peer networks, it has also become important to quantitatively measure the performance of each protocol and measure it up against available alternatives. This paper aims at providing an overview of the simulation scenario as far as peer to peer networks are concerned, and provides some quantitative data on the performance of some protocols. The paper then elaborates on the need for a generic network simulator for peer to peer networks and describe how such a simulator can be built, which is our work in progress. Such a network simulator would provide performance metrics for experimental P2P networks with minimal effort, leading to fast prototyping of newer protocols.

1 Introduction

A peer-to-peer computer network [10], more commonly known as a P2P network, is a network that primarily relies on the users of the network themselves for the resources required to maintain the network. This differs greatly from the more commonly used network topology, which relies on powerful centralized servers to maintain itself.

In a P2P network, all the nodes that are part of the network contribute bandwidth and computing power. Such networks are thus far more scalable and efficient, as every node added means more resources. Hence, the strength of a P2P network is its size. The most popular application for a P2P network is that of file sharing [5]. This involves distribution of digital content amongst all members of the network. The second most popular application is VoIP, or Voice over Internet Protocol . Such P2P networks allow transmission of high-quality audio over the participating nodes. We shall now take a look at the broad types of P2P networks that have evolved over the course of time.

2 Classification

A pure P2P network does not have the notion of clients or servers, but only comprise of peer nodes. All peer nodes are equal in status and each node functions both as as a client and server simultaneously to other nodes on the network. However, most P2P networks today have some element of the client-server architecture in them. In fact, most networks use servers that allow nodes to connect to the P2P network and search its contents.

The earliest form of a P2P network was Usenet. Usenet can be viewed as a P2P network of news servers scattered across the globe. These servers exchange messages with each other on a regular basis, mostly over the UUCP (Unix to Unix CoPy) protocol. However, each of these news servers acted as a server to the clients that it served. The servers merely synchronized with other news servers and notified them when one of their clients posted a message.

Popular P2P networks such as IRC and Napster also use a hybrid setup, much like Usenet did. The task of locating a file is performed using a server-client architecture, while the actual data, once found, is transferred from one or more peers to another.

To summarize, in pure P2P networks, all peers act as servers, merging the roles of the client and the server. There is no central server that manages the network. On the other hand, in a Hybrid P2P model, there is a central server that

keeps track of peers and responds to requests for information on them. Peers are responsible for contributing all resources required by the network, but they also update the server about their location and capabilities.

2.1 Unstructured P2P

All pure P2P networks are mostly unstructured in nature, which means that links between nodes in the network are formed arbitrarily. When a new node wants to join the network, it must know the address of at-least one node that is already connected and copy over all the existing links of that node. Links can then be formed with other unknown nodes over time using the simple Friend-of-a-Friend logic.

In this kind of a network, whenever a node wants to find a piece of data, the query has to be flooded across the network in order to find as many peers as possible. This works for data that is popular but not for data that is rare. Also, the size of the P2P which was actualy one of the greatest strengths of a P2P network works against effective searching. Since a single query has to be sent to each and every peer on the network to be entirely successful, the network is quickly flooded with a high amount of query traffic.

Such networks, therefore, are known to have very poor search efficiency. However, once the data is actually found, things move fast, since the exact addresses of all peers ready to send the required data is available. An attempt to establish a direct connection to each of these peers is made, and the data is then transferred through conventional transfer protocols (TCP, UDP, FTP etc) with the peer sending the data acting as the server, and the peer receiving the data acting as the client.

2.2 Structured P2P

Structured P2P networks attempt to overcome the shortcomings of Unstructured P2P networks by maintaining Distributed Hash Tables corresponding to the state and connectivity of the network. One peer is usually allocated a section of the network, and that peer is responsible for managing that specific part of content on the network.

Every peer that is part of the network, first applies hash functions to all its shared content and reports the values to the nearest peer that is responsible for that content. Modern protocols also specify metadata that must be attached to that report, such as the Album or Artist for an audio file. A global protocol determines which peer is responsible for a given data-set. This way, we ensure consistency in the two cases when a peer is searching for data, and when a peer is reporting presence of shared data. It is easy to see that Structured P2P networks are hybrid by nature, since certain peers are given more importance than others; and that some elements of the client-server architecture is borrowed.

3 Pure P2P Networks

We will digress a little to explain what pure P2P networks actually are and why we do not find any practical implementations of such networks.

A pure P2P network must make no assumptions during connection establishment, and must treat all members of the network equally. This would work in the ideal case where all members of the network possess equal resources (bandwidth, processing power etc). However, we find that in the real world, such a situation is far from plausible. Hence, we must come to rely upon nodes that possess greater resources to *help* nodes with lesser resources in connections establishment and content searches. Once this happens, you partly rely on the client-server architecture, thus making the peer network *impure*.

However, hybrid networks are still far more efficient at mass data transfer that conventional networks. Even though certain nodes are expected to contribute a greater share of their resources towards the network, the network will never be left without any of these *greater* nodes, since the network will simply pick the nodes having the largest resource pool amongst the available nodes.

Another issue regarding the implementation of pure P2P networks is that of bootstrapping. In order to join a P2P network, a node must already have some information about at-least a single node already part of the network. After connecting to it, this node can be said to be biased towards the node that introduced it, since further connections to the network will be based on what connections this initial node already has. However, with the introduction of small world networks, this problem can be eliminated by making at-least n passes at connections, where n is determined by the size of the network and the application of the small world network algorithm.

4 Advantages

The most important advantage of a P2P network is the sharing of resources. Since all peers contribute to the resources of the network, there is theoretically no maximum limit on the bandwidth and computing power available to sustain the network. Under a P2P model, the network can grow to an infinite size, since every peer added will bring along with it the additional resources required to maintain itself. This is a stark contrast to the classical client-server situation where adding more clients means slower data transfer given a fixed number of servers.

Another significant advantage in P2P networks is that there is no single point of failure in the entire system. The usage of multiple servers in a client-server setup may make a difference in sustaining the model, but P2P networks are immune to event the most wide-scale failures.

On a little investigation, one can infer that P2P networks are just a scaled down version of the Internet itself. The Internet was originally designed to be a completely distributed system, with no central authority or server maintaining it (RFC-I). In fact, several P2P networks are actually inspired from one of the core services of the Internet: The DNS or Domain Name System. DNS is a great example of how P2P networks are capable of scaling to handle any volume.

5 Real-world Protocols

There are several implementations of P2P networks existing today, and in wide use by the general Internet public. Most of them are minor variations over some major protocols that we will discuss here.

5.1 Napster

The one that started it all. Napster was the first widely used P2P network deployed over the Internet. Although Napster followed a largely server-client architecture, using decentralization only for the actual transfer itself, it sparked off the development of P2P networks that were more decentralized.

In fact, the primary driving force behind the development of networks such as Gnutella and Overnet was the fact that the RIAA was able to shut down Napster and its services almost overnight. Napster's original protocol still remains closed source, with its services now being offered in a pay-per-download avatar by Napster Inc.

5.2 FastTrack, eDonkey and Overnet

The FastTrack protocol was originally used by the Morpheus P2P client, and later on by programs like Kazaa, Grokster and iMesh. FastTrack is the most popular protocol for sharing small files like MP3 s because of its ability to resume interrupted downloads, and to simultaneously download segments of one file from multiple peers.

FastTrack uses the concept of Supernodes. The supernode functionality is built into the client, any peer with sufficient bandwidth and computing resources will automatically become a supernode, effectively acting as a temporary indexing server for other, slower peers.

To connect to the network, a client looks up a list of master supernode IP addresses that are stored in the program. These master supernodes will then return a list of currently active supernodes that are logically near to the client. The client caches the response and then connects to one of the returned supernodes. This supernode is called the client's upstream. The client uploads a list of files that it intends to share to its upstream, and also sends any search requests to it. The supernode forwards search requests that it cannot satisfactorily resolve to its neighboring supernodes, similar to the DNS system.

FastTrack uses a hash function called UUHash, which is a lightweight algorithm that is capable of hashing large files in a very short period of time even on slow computers. However, UUHash only hashes a fraction of the file itself, making it very easy to create a hash collision, allowing large chunks of data to be altered without changing the checksum.

UUHash will hash the first 300 kilobytes using MD5 and then apply a smallhash function (identical to the CRC32 checksum used by PNG) to 300 KB blocks at file offsets 2^n MB with *n* being an integer incremented from 0 until the offset reaches end of file. Finally, the last 300 KB of the file are hashed. If the last 300 KB of the file overlap with the last block of the 2^n sequence this block is ignored in favor of the file end block. The 128 bit MD5 hash and the 32 bit smallhash are then concatenated yielding the 160 bit hash used to identify files on the FastTrack network. The actual hash used on the FastTrack network is a concatenation of 128 bit MD5 of the first 300 KB of the file and a sparse 32 bit smallhash calculated in the way described above. The resulting 160 bits when encoded using Base64 become the UUHash.

FastTrack largely remains closed source, and open source programmers have succeeded in reverse-engineering the portion of the protocol dealing with client-supernode communication. The protocol followed by two supernodes for communication remains largely unknown.

Another protocol very similar to FastTrack is the eDonkey [6] network. eDonkey overcomes the hashing problems of FastTrack by using the much more robust MD4 algorithm instead of UUHash. As eDonkey became popular however, there was an increasing strain on the the supernodes running the network, defeating the purpose of a P2P network itself. Any supernode on the network was almost always made to suffer heavy traffic and was also vulnerable to attacks.

The successor to eDonkey is often called the Overnet, and is implemented by the eMule client. Overnet tries to overcome heavy reliance on the supernodes, by using a more decentralized protocol called Kademlia [9]. In this protocol, every peer attached to the network is responsible for some of the content. All nodes connect to each other using UDP, and Kademlia is a type of Distributed Hash Table. Information about content is stored in values , while each element of content is identifies by a key . When a peer needs to search for some data, it first calculates the corresponding key, and ask all its neighboring nodes for the location of it. The protocol, by design ensures that the number of nodes contacted during a search is only marginally dependent on the size of the network itself; only the closest nodes are queried.

5.3 Gnutella

Gnutella [1] uses a broadcast model to conduct queries, and was developed to make a fully distributed alternative to semi-structured systems like FastTrack. The networks' growing popularity only goes to show the scalable nature of the network.

Gnutella uses IP as its underlying network service, while the communication between hosts is specified in a form of application level protocol supporting following types of messages:

- **Ping,** A request for a certain host to announce itself.
- **Pong,** Reply to a Ping message. It contains the IP and port of the responding host and number and size of files shared.
- **Query,** A search request. It contains a search string and the minimum speed requirements of the responding host.
- **Query hit,** Reply to a Query message. It contains the IP and port and speed of the responding host, the number of matching files found and their indexed result set.
- **Push,** File-download requests for servants behind a firewall.

The bootstrap for a Gnutella client is very similar to that of a Freenet node. A Gnutella client must first connect to at-least one other node, the address of which it determines in one of several ways; such as preset addresses shipped with the software, using continuously updated web caches (called GWebCaches), UDP host cache entries or even IRC. Once connected, the client will request for a list of working addresses. The client then connects to all those addresses sequentially until it reaches a particular threshold value. Active addresses that the client has not connected to will be cached locally, and invalid (or unavailable) addresses will be discarded.

When the user initiates a search request, the client forwards the request to all the nodes it is connected to, which in turn forward the same to the nodes they are connected to. This happens until the query has hopped for a predetermined amount of time (TTL). Results of a particular search returns the addresses of all nodes that have the particular file. The client may not initiate a direct connection with those addresses and retrieve the file via UDP. The broadcast is breadth-first search, which guarantees that the optimal shortest path to the data will always be found. But, the price paid for a quick result is a large expenditure of effort to search the network. Gnutella makes a trade-off of much greater search effort in return for optimal paths and better worst-case performance.

One disadvantage of Gnutella and other similar systems is that the TTL effectively segments the Gnutella network into subnets, imposing on each user a virtual horizon beyond which their messages cannot reach. If on the other hand the TTL is removed, the network would be swamped with requests. This suggests that the Gnutella network is faced with a scalability problem.

If the client that sent the query is behind a firewall, such that no external host can contact it, the client sends a push request to a push proxy that asks the node that has the file to send it to the client, via TCP or UDP. In the original Gnutella specification; search results, push requests and file transfers were transferred in the exact path through which it was sent. This was found to be quite unnecessary, and the revised Gnutella protocol increased the overhead of the search queries to include the address of the source peer, so that direct connections are now feasible, and paths can be forgotten.

In practice however, it was found that searching the Gnutella network was often unreliable. Since each node is a regular computer, it may disconnect or connect at any time, making the entire network extremely dynamic and unstable. To avoid such bottlenecks, Gnutella implemented a tiered system of ultrapeers and leaves. Any new node that has just been connected to the network is attached as a leaf at the edge of the network, while nodes with considerably higher bandwidth and routing facilities were assigned roles of ultrapeers. Leaves never route any search queries, but only respond to direct connection or file listing requests. This revision made Gnutella a lot more faster and reliable.

5.4 BitTorrent

BitTorrent [7] is a file-distribution protocol that is based on P2P networks. The free software implementation of the protocol is also known by the same name. The protocol was originally designed by Bram Cohen. BitTorrent is generally considered to be the most successful specification of a P2P file sharing network, and is in wide use for distribution of digital content.

BitTorrent is designed to use the strengths of P2P networks to distribute large amounts of data without the burden of an equivalent consumption of server resources. The primary driving force behind BitTorrent is the fact that peers start uploading as soon as they begin downloading some content.

To share a file through BitTorrent, a peer must create what is known as a Torrent. This is a small file that defines the data to be distributed and some other information, which we will look into now.

The torrent file is split into 2 major sections. The first section, known as announce specifies a URL that identifies a host computer that will co-ordinate the file distribution.

This host is also known as the tracker, and the URL as the tracker URL. Although a host computer is required to coordinate the file distribution, it must be noted that the bandwidth and amount of resources required by the tracker very less in comparison to the size of the data that is actually transferred. Additionally, hardly any peers hosts a tracker on its own. Several BitTorrent communities have sprung up that provide trackers for popular torrents indexed by it. A tracker s primary role is to maintain a list of peers that are currently participating in the torrent.

The second section, known as info contains metadata of the file(s) being distributed. This includes the (suggested) names for the file(s), the piece-length used and an SHA-1 hash code for each of the pieces. This data is used by a receiving peer to check the integrity of the data being received.

Once a torrent file has been prepared, it must be hosted on a web server where people can find and download it. Nowadays, several BitTorrent communities allow users to upload their own torrent file, while they offer a tracker and some web space to host the torrent file itself. These torrent files are then indexed so that prospective peers can find it easily. Once a peer obtains a torrent file, the BitTorrent client parses the torrent file and contacts the tracker (at the tracker URL) to obtain information on all the peers that are supplying the data. Once this data is received, the peer is now part of the network and begins the transfer of data.

Some terms that are commonly used in the BitTorrent protocol are defined herein:

- **Seed,** a peer that offers the complete version of a file to all peers that request it. The computer that started the file distribution is called the *initial seed.* As soon as a peer completes the download of a file, it becomes a seed in the network.
- **Leech,** a peer that has just begun the transfer of data or has a partial version of a file. Note that leechers also upload data to other peers.
- **Stuck,** a state of the BitTorrent network when there are no seeders left, AND all the peers collectively don't have the complete file. This may happen if the initial seeder leaves the network before at-least one more peer has the complete file, or when all seeders leave the network at a point where all peers collectively don't have the complete file. At this point several peers have incomplete versions of the file (0 to 99%), and a reseed is required to start the network again. An interesting point to note here is that a BitTorrent network need not be stuck if there are no seeds. For example, if the initial seed leaves the network when one peer has the first 50% of the file, while another peer has the other 50%, these two peers will simply continue exchanging portions of data until they both become seeds!
- **Distributed Copies,** the effective number of copies of the file available on the network.
- **Swarm,** a BitTorrent network distributing a particular file (or set of files).
- **Choked,** the state of a peer with respect to another peer when it cannot send any data to it. This usually happens due to bandwidth constraints or in cases where the peer has set a maximum number of peers it can transmit to simultaneously.
- **Interested,** the state of a peer with respect to another peer when it is able to transmit data to it.
- **Snubbed,** the state of a peer when it has not received any data from another peer in 60 seconds. A snubbed connection is an indication of a choked peer.

This leads to some interesting questions that can be raised with respect to the bittorrent network:

- *What happens when a tracker goes down while a transfer is going on?*
- *How fair is BitTorrent? (Share Rating and Tit-For-Tat)*
- *Can performance be improved if rarest data is transferred first?*
- *Can there be multiple trackers or trackerless torrents?*
- *Can the database be distributed? (DHT)*
- *Do concepts like anti-snubbing and optimistic unchoking exist?*

6 Academic Protocols

Unlike the networks discussed so far, these academic protocols are designed to solve the *lookup problem.* Not all of these protocols have practical real-world implementations, but are useful because they solve the problem of finding data very effectively, to be more precise, in a logarithmic number of routing hops. Since these protocols overcome the restricting limitations of the protocols discussed earlier, they are often known as 2nd generation P2P systems [8]. In the future, such lookup algorithms can possibly be *plugged in* to real-world P2P networks in order to boost performance significantly. Not only are these networks applicable to P2P networks, but to any decentralized system such as network storage and web caching, in general. We will briefly discuss a few of these protocols.

6.1 Chord

Chord focusses on models in which nodes join and depart in a well-behaved fashion, and allows maintenance to happen only at the time of arrival and departure. In a Chord system, data is normally found by associating a key with each data item, which is then stored as a pair at the node to which the key maps. The Chord protocol then makes all peers of the network *agree* upon which node actually contains the data item in $O(logN)$ messages where the size of the network is *N*.

Chord, by itself, only provides an algorithm for all nodes to execute, but doesn't say anything about how the data is actually stored. That is the role of DHash [3], also known as a *distributed hash table*, which is a layer built on top of Chord to handle the data storage.

The Chord protocol deliberately avoids TCP as its transport, since it was found to be very inadequate in distributed environments. Chord/DHash instead implements a custom transport layer optimized for peer to peer systems. Chord also uses the Vivaldi coordinate system to predict round-trip times between nodes.

Chord currently provides a single research implementation for Linux and BSD systems in C++ using the SFS (Self certifying file-system) library.

6.2 CAN

CAN (Content Addressable Network) [11] is essentially another distributed system that provides hash-table functionality on an internet-like scale. CAN, unlike Chord, does not make any assumptions on the structure of the network and the timing of data, and therefore, can be deemed to be more fault-tolerant. CAN uses a virtual multi-dimensional cartesian coordinate space on a multi-torus. The entire coordinate space is then dynamically partitioned among all the members of the network such that every peer possessed its own *distinct* zone within the system. As discussed before, this is an attempt to make the P2P system as pure as possible.

Every CAN peer maintains a routing table that holds the IP addressed and corresponding coordinates of every immediate neighbor. The peer then routes a message to its destination by using a simple *greedy* algorithm, i.e. passing the message to a neighbor peer that is closest to the destination coordinates. Consequently, we find that there are cases where the message is not routed along the shortest path.

CAN provides a bare-bones reference implementation in the paper that described it.

6.3 Pastry

Pastry [12] is an overlay and routing network for the implementation of a distributed hash table similar to Chord. Every piece of data is associated with a key, and then stored in a redundant fashion across the peer network. A node looking to join a Pastry network simply bootstraps by querying a single node already part of the network.

Although Pastry's DHT implementation is almost identical to that of other DHT's, Pastry differs in the concept of a routing overlay built on top of the DHT concept. Hence, Pastry can be as reliable and scalable as CAN, but still reduce the overall cost of routing a packet from one node to another. Pastry achieves this by using several routing metrics supplied by established external programs like ping or traceroute. This allows for dynamic routing decisions based on a variety of metrics like shortest hop count, lowest latency, highest bandwidth, or even a combination of them.

Two applications based on Pastry are PAST [4]: a distributed file system, and SCRIBE: a publishing system.

6.4 Tapestry and Chimera

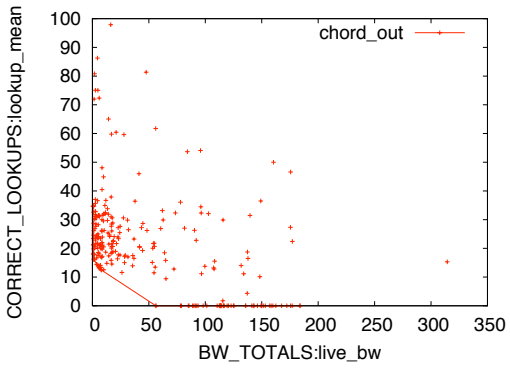
Tapestry [13] is a distributed hash table which provides a decentralized object location, routing, and multicasting infrastructure for distributed applications. Tapestry is fundamentally composed of a P2P overlay network that offers location-aware routing to nearby resources. Tapestry constructs locally-optimal routing tables for every node that joins it on initialization and therefore reduced routing stretch. Tapestry is used in several real-world applications such as Ocean Store (PlanetLab's distributed storage utility) and SpamWatch (A decentralized spam filter)

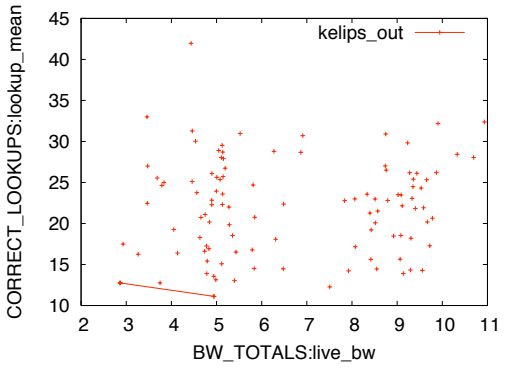
A new variant of the Tapestry called Chimera is currently being developed. Chimera is a light-weight C implementation of a "next-generation" structured overlay that provides similar functionality as prefix-routing protocols Tapestry and Pastry. Chimera gains simplicity and robustness from its use of Pastry's leafsets, and efficient routing from Tapestry's locality algorithms. In addition to these properties, Chimera also provides efficient detection of node and network failures, and reroutes messages around them to maintain connectivity and throughput.

7 Quantitative analysis

7.1 Sample Results

We tried to simulate the protocols chord, kelips and tapestry on two distinct topologies using MIT's P2PSim. The first network comprised of just seven nodes while the second had a 1024 node matrix. The results were obtained with lookup vs bandwidth generated as the plot shows. We ran this simulation three times over for each topology. The





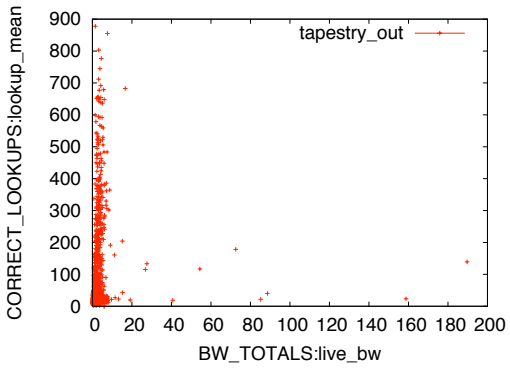
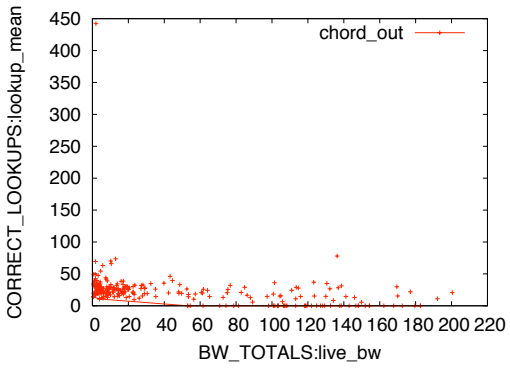
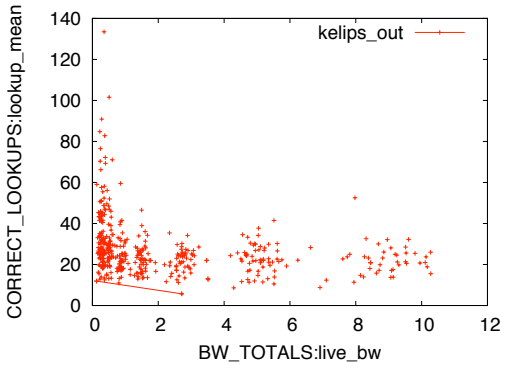


Figure 1. Simulation Run 1





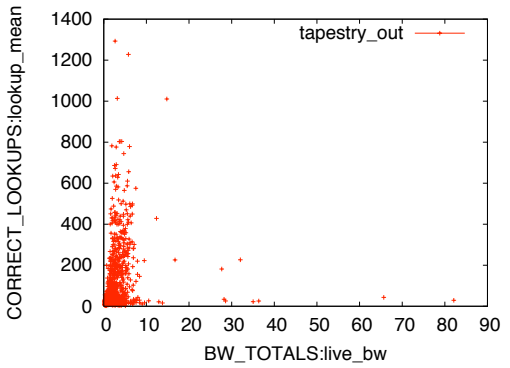


Figure 2. Simulation Run 2

simulator generates the results keeping in mind the network latency, which varies every-time the simulation is run.

Graph 1 and Graph 2 are the results of the first and second simulation runs on the seven node topology. It is observed that 1 was simulated with greater lookup times, in a more conjusted environment than 2. Kelips appears to show maximum vulnerability towards network congession while Tapestry was most consistent. Performance evaluation shows different results however. Since we are plotting lookup mean vs bandwidth, it is favourable to have maximum points towards the bottom right corner: i.e. higher bandwidth with minimal lookup mean. Whereas tapestry might exhibit lesser lookup mean than kelips, the bandwidth available to each node is very minimal. Chord performs best in this category with maximum points with low lookup times and relatively high live bandwidth.

Our test with 1024 nodes further projected this obtained result. Kelips, though provided with the best peak bandwidth, failed to meet consistency in lookup. Tapesrty on the other hand had very poor bandwidth and life time for each node. Chord behaved in a more moderate and consistent manner.

8 Simulation issues

In trying to simulate real-world peer networks, we came across several simulators but found that none of them could actually simulate both 1st and 2nd generation peer network protocols in the same environment. Most simulators were very specific to a certain protocol [2], for example there were several simulators pertaining to the simulation of Gnutella networks. We found that the plethora of simulation tools available were inadequate when it came to comparing two real-world peer networks.

Three simulators, however, came very close to what we were looking for. P2PSim from MIT came with implementations of 6 academic protocols: Chord, Accordion, Koorde, Kelips, Tapestry and Kademlia, but none of the 1st generation protocols. On a similar note, PlanetSim came with Chord and Symphony implementations; while GTNetS did not provide any P2P implementation but a rather large and generic framework to build upon.

However, none of these simulators had very good documentation on extending and implementing new protocols, and it was found difficult to extend the simulator to support real-world networks or perform comparative studies of overlay networks. Hence, the idea of a new modular peer network oriented simulator was floated, which would overcome the drawbacks of these simulators by providing an abstract, easy to use interface to build protocols on. We describe the proposed simulator briefly in the next section.

9 Proposed P2P simulator

As described previously, there is a need for a generic and modular network simulator geared towards peer to peer networks. We describe here our work in progress for such a modular simulator.

9.1 Architecture

The *defacto* standard for most network simulators is the event based model. Our simulator will not differ in this aspect and will report every event (creation of packets, losses, lookup successes and failures etc.) along with the timestamp.

What will make the simulator unique is modularity. The simulator will be built as a very loose coupling of abstract objects, which should make simulation of overlay networks a lot easier. For example, one may want to simulate a Chord network with a data storage mechanism other than DHash, or simulate traditional BitTorrent file exchange using Tapestry-style lookups. With such a modular architecture, it will be possible to mix and match different lookup protocols, their data storage mechanisms and transport layers.

Another aspect of the simulator which will differentiate it from the rest is the introduction of fault models. The simulator will be designed to introduce failures of different kinds at intervals as determined by a fault model chosen by the user. The simulator will also focus on emulating Distributed Denial of Services (DDoS) attacks across the network being simulated.

9.2 Input

An XML representation for the various input parameters such as topology, fault models and simulation characteristics will be designed, as it was felt that the text-based input in P2Psim was largely inadequate. The NED Language invented by the Omnet++ community is more powerful, and a similar implementation in plain XML is proposed for use in our simulator.

9.3 Diagnostics

Undoubtedly, the most important part of a simulation is the results. Our simulators will closely match the output of several existing simulators by providing a verbose event log during the simulation which will then be parsed into an XML output format on completion. The output can then further be converted to more appropriate formats such as GNUPlot or Matlab primitives for analysis. The simulator will favor verbosity over performance.

9.4 Implementation

One of the most important requirements of a generic P2P simulator such as ours is the ability to experiment with new protocols and networks. The proposed simulator will be built in Python, a highly flexible, cross-platform, easy to use language. Developing a core infrastructure that defines basic network objects and their implementations will go a long way in making the process of implementing new protocols as easy as possible. The simulator will strive to ensure that you spend lesser time coding, and more time in actually improving the protocol based on simulation results by allowing rapid prototyping. The simulator will be bundled with both a command-line and GUI interface for enhanced usability.

The simulator will be developed roughly over the following stages of development:

- Implementation of the core library with basic network protocols and general packet delivery algorithms
- Implementation of a set of lookup protocols, data storage mechanisms, and transport layers on the basis of abstract objects defined in the core library
- Coding simulation scripts and input parameters allowing usage of these objects in a simulation environment
- Implementation of diagnostics modules for parsing of output in several formats
- Introducing fault models and DDoS emulation
- Build GUI front-end and package final simulator

10 Conclusion

The presented survey provides a good idea of current state of peer to peer networks and simulation methods. The paper describes the operation of various unstructured and structured networks that are in wide use today, and performs a quantitative analysis of the 2nd generation DHT based protocols. The paper also look into the fallacies of peer network simulators available today, and gives a brief outline of a proposed modular simulator that should help researchers in rapidly prototyping and improving peer network protocols. To the best of our knowledge, there are no generic simulators available that make the analysis of real-world peer networks such as Gnutella, BitTorrent and FastTrack possible in a single environment. We hope that the new modular simulator will offer a unique tool to researchers by providing a suitable platform on which both 1st and 2nd generation peer network protocols as well as newer experimental ones can be evaluated.

References

[1] K. Aberer and M. Hauswirth. Peer-to-peer information systems: concepts and models, state-of-the-art, and future systems. In *International Conference on Data Engineering*, 2002.

[2] A. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent network's performance mechanisms. In *Proceedings of the IEEE InfoCom*, 2006.

[3] J. Cates. Robust and efficient data management for a distributed hash table. Master's thesis, Massachusetts Institute of Technology, 2003.

[4] P. Druschel and A. Rowstron. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *18th ACM Symposium on Operating Systems Principles*, pages 188–201, October 2001.

[5] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *Proceedings of the IEEE InfoCom*, 2003.

[6] S. B. Handurukande, A.-M. Kermarrec, F. L. Fessant, L. Massouli, and S. Patarin. Peer sharing behaviour in the edonkey network, and implications for the design of serverless file sharing systems. In *Proceedings of the 2006 EuroSys conference*, pages 359–37, 2006.

[7] A. Legout. Understanding bittorrent: An experimental perspective. Technical report, Institut Eurecom, 2005.

[8] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In *Principles of Distributed Computing*, 2002.

[9] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *1st Intl. Workshop on Peer-to-Peer Systems*, 2002.

[10] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. Technical report, HP Laboratories, 2002.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *ACM Special Interest Group on Data Communications*, 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[13] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications*, 22(1):41–53, 2004.