HodgePodge: A Scalable and Secure Peer-to-Peer Protocol

Steve Webb

Mini-Project 1 – “Secure Broadcast in a Peer-to-Peer System”
Advisor: Brian Cooper
CS7001
1 Introduction

Traditional networking systems are typically characterized by the client-server networking paradigm in which less powerful nodes (clients) rely on more powerful nodes (servers) to accomplish their intended task. The most obvious example of this approach is evident in web browsing where a web browser accesses a web server in order to obtain information. However, over the past couple years, a new peer-to-peer (P2P) networking paradigm has emerged, and unlike the traditional client-server model of networking, the P2P approach leverages the computing power and resources of each participating node. Thus, rather than making the distinction between clients and servers, P2P nodes are often referred to as servents (SERVer cliENTS), indicating their potential role as both a client and a server [1].

P2P networking systems offer a number of benefits, but they also introduce a number of important research topics. Currently, one of the primary areas of interest in P2P networking deals with searching the P2P infrastructure in an efficient and secure manner. In a client-server environment, clients are able to contact a centralized server in order to obtain the information they seek. However, the analogous situation in a P2P environment is much more complicated because a central authority does not exist. In order to obtain information in a P2P system, servents must contact each other in a distributed fashion until they find their desired results. As a result, the success of a servent’s request for information is ultimately determined by the robustness of the search mechanism used in the underlying P2P system.

A great deal of research has been done to show that current P2P searching techniques are inefficient and do not scale well [2, 3, 4, 5, 6, 7, 8]. Many of these search mechanisms rely on flooding, whereby servents broadcast search requests to each of their neighbors in order to send the request to every node on the network. Unfortunately, flooding has a number of flaws. First of all, flooding requires all servents in a P2P network to process every search request, regardless of whether or not they can respond with results. Thus, both processing power and network
resources are wasted. Additionally, servents share neighbors with other servents (i.e. a servent has many paths leading to it) so some servents will inevitably receive the same requests repeatedly as a single search request propagates throughout the network. Another concern arises if malicious peers participate in the network. Flooding assumes that servents will forward requests to each of their neighbors, but malicious peers might choose not to forward a request in order to undermine the success of the search.

In addition to showing the problems with current P2P search methods, research has also provided a number of useful techniques and ideas for improving P2P search mechanisms [2, 3, 4, 5, 6]. This paper presents a new P2P protocol called HodgePodge which incorporates the best contributions from previous research along with enhancements and new contributions. The remainder of this paper is organized in the following manner. Section 2 explains the related work done in this particular area of P2P networking. Section 3 identifies the inherent flaws in current searching techniques. Section 4 describes how the components of HodgePodge solve current P2P searching problems. Section 5 gives conclusions and aspirations for future work.

2 Related Work

For the past few years, a number of studies have been conducted in an attempt to improve the effectiveness of P2P searching. One unique approach is a more structured P2P environment based on distributed hash tables (DHT). This DHT-based idea has been adopted by a number of new P2P protocols such as CAN [9], Chord [10], Pastry [11], and Tapestry [12]. Due to the additional structure imposed by these protocols, searching efficiency is enhanced. However, the protocol proposed in this paper (HodgePodge) is a much less structured protocol, similar to Gnutella v0.4 [1] and KaZaA [13].

Beverly Yang and Hector Garcia-Molina illustrate the benefits of incorporating Super-peers into P2P networks [2]. Their paper makes the observation that each participating servent in
A P2P network has a varying amount of resources (e.g. processing power, bandwidth, etc.). Thus, they propose electing the more powerful servents to act as Super-peers which then become the gateway by which less powerful servents communicate with the P2P network. By creating this hierarchy, Yang and Garcia-Molina illustrate that less structured protocols, such as Gnutella v0.4 [1], can become much more scalable and that searching efficiency can be increased in the process.

Beverly Yang and Hector Garcia-Molina provide a few improved search techniques: Iterative Deepening, Directed BFS, and Local Indices [3]. Of these three approaches, Local Indices has the most promise because it introduces the notion of a servent having knowledge of its neighbors’ data for routing purposes. This idea is extended by Arturo Crespo and Hector Garcia-Molina as they propose the idea of a compound Routing Index (CRI) [6]. The CRI is very useful in making routing decisions for a query because it essentially gives an estimate for the likelihood that a given neighbor will be able to respond to the query.

Vana Kalogeraki et al. also discuss a few improved search techniques: Modified Random BFS and Intelligent Search [5]. Intelligent Search is very similar to the routing indices idea proposed by Crespo and Garcia-Molina [6] in that both schemes rely on some heuristic for determining which neighbors should receive a given query. Another interesting idea proposed in the Intelligent Search mechanism is the notion of comparing two queries based on their cosine similarity. This idea can be used to reduce the amount of information each servent must maintain when routing queries.

Qin Lv et al. explain a number of issues relating to P2P systems such as content availability, replication strategies, the inefficiency of flooding, and two new search schemes: Expanding Ring and Random Walk [4]. The Random Walk concept utilizes a “walker” to go from servent to servent, collecting results. Lv et al. show that this searching mechanism is significantly more efficient than flooding and only marginally increases the response time. They
also illustrate that square-root replication of data provides an optimal search size and that this replication ratio can be attained with path replication and Random Walk.

Jordan Ritter attempts to quantify the reason Gnutella’s searching strategy will never scale [7], and Serguei Osokine provides a similar analysis that encompasses additional topics such as Super-peers and Mutual Index Caching [8]. Both of these papers serve to illustrate the inefficient nature of P2P searching based on flooding.

3 P2P Searching Flaws

Currently, most P2P searching in unstructured P2P systems is accomplished by flooding. If servent A wants to search the network, it begins by sending a search request to all of its neighbors. Upon receiving this request, servent A’s neighbors respond with all of their results and then forward the search request to all of their neighbors. Eventually, the search request reaches all of the neighbors in the network, and servent A receives all of the available results.

Although this approach is very simple and easy to implement, it has a number of problems. First, regardless of whether or not a servent can return results for a given request, the servent must expend resources to forward the request to its neighbors [3, 4]. Thus, many servents will be forced to waste resources despite not being able to return results for a given request. To illustrate this problem, let us assume the P2P network has 100 servents when servent A sends a request for a file named X. Further, assume that only 10 servents in the network are able to return results for file X. In this scenario, 89 servents (100 – 10 – servent A) are forced to process the request even though they are unable to return results. Obviously, some of these wasted resources are unavoidable because some servents are needed to route requests to the 10 servents that can return results. However, flooding-based systems are guaranteed to waste the maximum amount of resources.
The next problem with flooding is an artifact of the high connectivity typically found in P2P networks. If each servant only had one incoming connection and one outgoing connection, each flooded request would only arrive at each servant once. However, most servants in a P2P network have many connections; consequently, they share neighbors with other servants. Thus, during a flood-based search, the same request is being routed to certain servants numerous times [4]. As a result, additional resources are wasted as duplicate requests are discarded.

Another problem with flood-based searches involves security. In order for flooding to work, all servants must route the search request in an appropriate manner. If a malicious peer were to enter the P2P network, it could potentially disrupt the effectiveness of the search by refusing to propagate requests. As mentioned above, flooding introduces a large amount of redundant data due to the connectivity of the P2P network, but if a malicious peer positions itself appropriately (i.e. at a highly connected point in the network), there is no guarantee that the request will be delivered to all of the servants in the network. Additionally, as the number of malicious peers increases, the probability that the search will be disrupted also increases.

4 HodgePodge Specification

Due to the various limitations and inefficiencies involved with flood-based search mechanisms, this paper proposes a new search mechanism that incorporates the findings of previous research in addition to original contributions. However, rather than using this new search mechanism on top of an existing protocol, HodgePodge defines an entirely new protocol to help increase reliability, efficiency, and security. Thus, in addition to the search mechanism, this new protocol also includes a modified topology and a replication strategy. Each of these components contributes to the overall protocol design, and when taken as a whole, this new protocol provides a much more reliable P2P searching environment.
4.1 Topology

The underlying topology of the HodgePodge network is a two-level hierarchy that utilizes Super-peers. Each Super-peer has some number of connections to less powerful servants called leaf nodes (or leaves) and some number of connections to other Super-peers. The exact number of connections will vary depending on the resource limitations of the Super-peers, but [2] suggests that Super-peers should maintain as many connections as possible. The primary responsibility of a Super-peer is to route the traffic of its leaves (including its own traffic) to its neighboring Super-peers.

Yang and Garcia-Molina [2] give sufficient evidence that illustrates the benefits of Super-peer networks: efficiency, reliability, and scalability. In addition to the benefits listed by [2], Super-peer networks are also able to maintain compound Routing Indices in a much more efficient manner than unstructured P2P networks. Essentially, a Routing Index (RI) is information provided by a servant about the types of requests for which it can provide results [6]. Since a Super-peer is already responsible for routing traffic for some number of less powerful servants (its leaves), it is logical to have the Super-peer maintain information about the requests those servants are able to answer (the Routing Index). If the Super-peer maintains RIs for each of its leaves, it will be able to make intelligent routing decisions upon receiving a search request. If leaf A is unable to return results for a given request, the Super-peer will not send the request to leaf A. Thus, resources are not unnecessarily wasted.

In addition to using a Super-peer network, HodgePodge also incorporates the idea of k-redundancy as it is explained in [2]. The basic concept is that instead of relying on a single Super-peer, leaves connect to k Super-peers which act together as a single “virtual” Super-peer. It is instructive to refer to a Super-peer and its leaves as a cluster. Thus, if the network consists of k-redundant Super-peers, each cluster will consist of k Super-peers along with the cluster’s leaves. From a leaf’s perspective, k-redundant Super-peers function the same way as traditional Super-peers. The leaf has k times as many connections (one for each of the k Super-peers), but
fundamentally, the network works in the same manner. However, with k-redundancy, the network is far more reliable [2]. In traditional Super-peer networks, if the Super-peer fails for any reason, the leaves in that Super-peer’s cluster become fragmented from the rest of the network until they are able to join a new cluster. In HodgePodge, all k Super-peers have to fail before a cluster’s leaves become fragmented from the network.

In addition to reliability, another very appealing aspect of k-redundant Super-peers is their load-balancing qualities. Since each of the k Super-peers is connected to each of the leaves, all incoming traffic can be split among each of the Super-peers in the cluster. Thus, the amount of traffic the k-redundant cluster can handle is almost k times more than a traditional Super-peer. Additionally, when the k-redundant cluster becomes overloaded, a sufficiently powerful leaf node can be promoted to a Super-peer, creating a k+1-redundant cluster. As a result, the k-redundant cluster is easily expandable if necessary.

K-redundant Super-peer clusters can be implemented using Unicast, Multicast, or Anycast. The Unicast approach is the traditional implementation discussed in [2]. Using this approach, when a leaf connects to a Super-peer, the Super-peer returns a list of the other Super-peers in the k-redundant cluster. The leaf then connects to each of the other Super-peers. Once the leaf has connected to all of the Super-peers in the cluster, it determines which of the Super-peers to use based on any number of algorithms (e.g. Round-Robin, Random, etc.). The Multicast approach is more elegant than the Unicast approach, but due to the limited support for IP Multicast, an application layer Multicast protocol such as Narada [14] must be used. In this approach, each of the k-redundant Super-peers is a member of a Multicast group. When a leaf or another cluster sends one of the Super-peers a request, it is sent to the Multicast group. As a result, no one outside the Multicast group knows anything about the members of the group; they simply know someone in the group will process their request. Also, unlike in the Unicast approach, this approach requires the Super-peers to decide amongst themselves who will process an incoming request. The Anycast approach is even more elegant than the Multicast approach. In
this approach, each of the k-redundant Super-peers is a member of an Anycast group. Then, just as in the Multicast approach, when a leaf or another cluster sends one of the Super-peers a request, it is sent to the address of the Anycast group. Anycast works well in this environment because the actual Super-peer chosen is unimportant; it is only important that a Super-peer receive a request, not which Super-peer receives it. The specifics of Anycast are beyond the scope of this paper, but more information can be found in [15].

4.2 Searching Mechanism

The searching mechanism HodgePodge uses is a modified version of the Random Walk approach described in [4]. HodgePodge sends an arbitrary number of walkers along various paths to find the desired content. A walker is essentially a smart-request that follows a single path as opposed to a flooded-request that follows all paths. The exact number of walkers depends on a number of factors such as the popularity of the content, the patience of the user, the desired number of results, etc.

Unlike the random walkers used in [4], the walkers HodgePodge uses are much more intelligent. Using the RI information maintained by each Super-peer (discussed above), a HodgePodge walker only walks down paths that are likely to return results. An example should clarify this point. Let us assume leaf A sends a request to one of its k-redundant Super-peers. When the Super-peer obtains this request, it looks at its RI information to determine which of its Super-peer neighbors can process the request. Once it knows which neighbors will return results, the Super-peer sends walkers to those neighbors. After the walkers arrive at these neighbors, the neighbors return their results for the request, and then they forward the walkers to their own neighbors that can process the request. This process continues until the request has generated enough results.

The previous example clarifies the Random Walk procedure, but it raises a couple issues. First, the walkers must have some mechanism that tells them when to stop walking. In [4], the
authors advocate using a “checking” method whereby the walker contacts the original requester to determine if it should continue walking. In theory, this is a good idea, but in practice it is not efficient or realistic. First, if the walker contacts the originator after each step (or even after every fourth step as proposed in [4]) it will generate unnecessary traffic. Additionally, these checking messages will have to be back-propagated to the originator, and then the originator’s response will have to be re-routed to the walker’s current position. This process takes time, and as a result, it could degrade the overall timeliness of the search. Instead of using the “checking” method proposed in [4], HodgePodge associates a user-defined number of desired results with each walker. Then, as the walker walks along the network, its desired number of results is decremented as it obtains results from the network. When the walker obtains its desired number of results, it stops walking.

Another important issue that arises in Random Walk is duplicate data caused by cycles in the network. HodgePodge attempts to solve this problem by associating a unique ID with all walkers involved with a given query. For example, if k walkers are used to obtain results for request X, each of the k walkers is given the same ID to denote its involvement with request X. Then, if a Super-peer receives a walker with an ID it has already seen, the Super-peer can choose to discard the duplicate walker, or the Super-peer can choose to send the walker along a different path (i.e. to a different neighbor than before). Thus, as the walkers are walking throughout the network, cycles can be detected and handled accordingly.

4.3 Data Replication

The importance of data replication in a P2P network has been emphasized in recent studies [4, 5]. This replication research makes the observation that the replication of data in a P2P network results in a reduction in the total amount of traffic because the workload becomes more evenly distributed throughout the network. As a result of this observation, HodgePodge
incorporates an active replication scheme to further improve the scalability and efficiency of its search model.

According to [4], the optimal replication strategy is the square-root replication scheme. This replication scheme essentially says that data should be replicated proportional to the square root of the number of requests for that data. Additionally, [4] explains that the Random Walk searching mechanism can be used to implement a “path replication” strategy which yields results that are very similar to that of square-root replication. In the “path replication” approach, the requested data is stored along every hop between the servant that requested the data and the servant that provided the data [4]. However, in HodgePodge a slightly different replication strategy is implemented. Initially, Super-peers cache pointers to the data requested by their neighbors. This caching process occurs as the Super-peers route results back through the network. As these requests become more popular, the Super-peers have the option of storing the actual data locally to increase the availability of the data while reducing the amount of traffic needed to access it. Overall, this scheme for data replication is just an additional way to help improve the performance of searching in HodgePodge. It is not as important as HodgePodge’s topology or search mechanism, but it does contribute to the protocol’s efficiency and reliability.

4.4 Security

The best way to evaluate the security of HodgePodge is to analyze some scenarios:

- Scenario 1: M malicious peers are leaves for some number of the k-redundant Super-peers.

In this scenario the threat posed by the malicious peers is quite trivial. Essentially, they only have two real avenues of attack. They could attempt to flood the Super-peers with illegitimate search requests, or they could provide the Super-peers with invalid RI information.

The first attack will only be mildly successful for a number of reasons. First, since k Super-peers are distributing the load, it is k times harder to disrupt the operation of the k-
redundant Super-peers. Additionally, if the Super-peers begin to receive an inordinate amount of traffic from a given leaf node, they will remove that leaf node from the cluster. Another point to remember is that unless the requests being used for the flood are unpopular, the flooding peer will inevitably become flooded by the backlash of results created by its flood of requests. However, assuming the flooding peer survives this backlash and is not removed from the cluster, this flooding attack could be used as an escalation-of-privileges attack. In HodgePodge (and traditional k-redundant Super-peer networks), capable leaf nodes are promoted to being a Super-peer if the load increases dramatically. Thus, if a malicious peer previously claimed itself as being capable, it could use a flood to increase the load on the cluster. This would in turn allow the malicious peer to become a Super-peer in the cluster. Specific attacks relating to this scenario are provided below.

The other attack in this scenario is a bit more effective, but ultimately, it will be dealt with in a similar fashion as the first attack. In the normal operation of the protocol, a fairly constant stream of updates for RI information will be received by the Super-peers. Thus, it is quite challenging to distinguish between legitimate RI traffic and invalid RI traffic. However, it is not impossible. First of all, if the RI traffic being generated is completely unreasonable (i.e. too much data in too short a time period), the Super-peers can remove the source from the cluster. Also, once search requests arrive, it will be easier to determine the legitimate leaves from the malicious ones. Typically, malicious peers that are sending invalid RI information will not be able to respond to the requests that correspond to that RI information, and as a result, the malicious peers can be identified. The problem with this approach is that some malicious peers will simply respond with a generic response regardless of the actual request, and that strategy requires more processing to detect.

- Scenario 2: M malicious peers form some number of M-redundant Super-peers.

In this scenario the threat posed by the malicious peers is more serious. When malicious peers participate as Super-peers in a k-redundant Super-peer both legitimate leaves and legitimate
Super-peers can become victims of an attack. The main attacks malicious peers can perform as Super-peers are as follows. The malicious peers can flood other Super-peers with illegitimate search requests, they can provide invalid RI information to other Super-peers, they can refuse to route the requests they receive from the leaf nodes in their cluster, and they can provide illegitimate search results to the leaves in their cluster.

The first attack involves the same type of flooding described above in Scenario 1. The only difference is that now the malicious peer is a Super-peer. Thus, the countermeasures provided by HodgePodge for this situation are very similar. Successful flooding is inherently difficult due to the k-redundant Super-peers. In order to fragment a cluster, all k Super-peers that comprise that cluster must be eliminated. Thus, accomplishing this task is sufficiently difficult to discourage the use of flooding as effective attack.

The next attack is also very similar to the RI attack described in Scenario 1. However, there are some slight differences. The amount of RI information that is distributed among the neighboring Super-peers is significantly greater than the amount of RI information exchanged between a leaf and a Super-peer. As a result, it is even more difficult for a legitimate Super-peer to detect a malicious peer’s invalid RI information. However, malicious peers should eventually be identified when they provide invalid data during a file transfer.

Another attack performed by a malicious Super-peer involves refusing to route leaf node requests. When a malicious peer becomes a Super-peer in a cluster, it obtains the ability to process the requests of the leaves in that cluster. Thus, if the malicious peer wanted to disrupt the search mechanism for its leaves, it could refuse to forward their requests. In doing so, the leaves would be unable to obtain any information from the network. To combat this attack, leaf nodes using HodgePodge must monitor the success of their search requests. If they are consistently unsuccessful, they must consider connecting to a different k-redundant Super-peer. Additionally, the leaf could give the Super-peer a bad rating (for trust purposes or performance purposes). The
concept of trust and the HodgePodge rating system have not been fully analyzed and are beyond the scope of this paper.

In the previous attack, instead of ignoring the requests of the cluster’s leaves, the malicious Super-peer could respond with illegitimate responses. These illegitimate responses would be hard to distinguish from legitimate responses so the leaf nodes would have to waste a significant amount of time and resources to discover their illegitimacy. Upon discovering the malicious activities of one of the Super-peers, a leaf node should connect to a different cluster.

5 Conclusions and Future Work

As the Peer-to-Peer networking paradigm continues to grow in popularity, the ability to efficiently and securely search P2P systems will become increasingly important. In the past, P2P systems have relied on flooding-based search mechanisms, but this paper has shown that these methods are inherently flawed. Consequently, this paper proposed HodgePodge, a new P2P protocol that offers a scalable and secure system that can be used for more efficient searches. By incorporating the best P2P techniques currently available, HodgePodge offers a significant improvement over existing P2P systems, and it should facilitate the continued research of efficient and secure P2P searching mechanisms.

In the future, HodgePodge should utilize a peer-rating system that can be used to introduce the concept of trust into the network. Then, by using these trust values, peers will be able to make better connection and routing decisions. Additionally, Osokine [8] explains an interesting topology called Mutual Index Caching. As HodgePodge evolves, new topologies such as the one proposed by Osokine should be tested in order to further refine the scalability of the protocol.
References


