Kudzu:
A Decentralized and Self-Organizing Peer-to-Peer File Transfer System

by
Sean K. Barker

Jeannie Albrecht, Advisor

Contents

1 Introduction 8
- 1.1 Goals . . . 10
- 1.2 Contributions . . . 10
- 1.3 Contents . . . 11

2 Background 12
- 2.1 Networking Paradigms . . . 12
- 2.2 P2P Paradigms . . . 13
  - 2.2.1 Napster . . . 13
  - 2.2.2 Kazaa . . . 14
  - 2.2.3 Gnutella . . . 15
  - 2.2.4 BitTorrent . . . 16
  - 2.2.5 DHTs . . . 18
- 2.3 Properties of P2P Networks . . . 19
  - 2.3.1 Scalability . . . 19
  - 2.3.2 Incentives . . . 19

4 Implementation: The Kudzu Client 40
4.1 Communication Framework 40

List of Figures

Abstract

The design of peer-to-peer systems presents di cult tradeo s between scalability, e ciency, and decentralization. An ideal P2P system should be able to scale to arbitrarily large network sizes and be able to accomplish its intended goal (whether searching or downloading) with a minimum amount of overhead. To this end, most P2P systems either possess some centralized components [illegible]

Chapter 1

Introduction

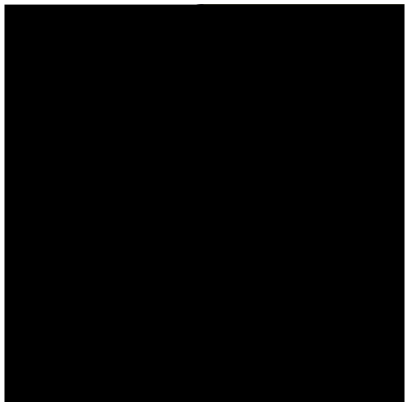
In the past decade, one of the greatest bene ciaries of increasing consumer broadband adoption has been the development of peer-to-peer (P2P) systems. The traditional model of online content consumption is based around dedicated providers such as corporate web servers that provide up-stream content to home users and other content consumers. In this model, providers are generally companies or technically savvy users, but the majority of Internet users do not s4372(o357(con)28(ten)28rs)-2irecterally

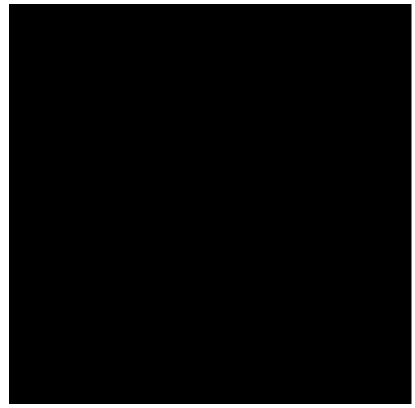
the e cacy of our design and draw conclusions about decentralized P2P systems of this type. In

Chapter 2

Background

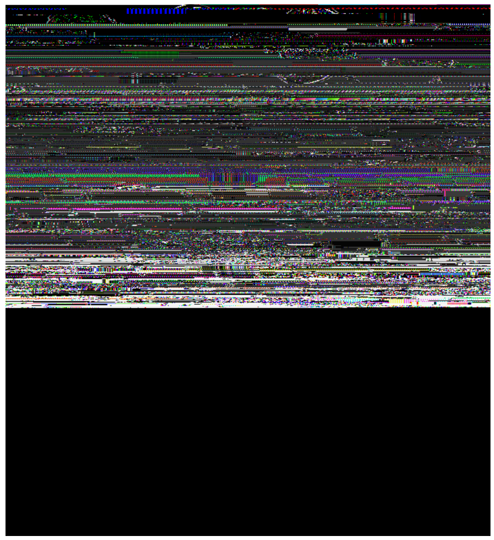
2.1 Networking Paradigms





2re112578(P2P)7578aradigmsMS

seem manageable at first glance, note that this means the total amount of traffic the network has to handle grows exponentially; each new node has to handle each new query, resulting in more and more bandwidth used as the network grows. An analysis of early Gnutella bandwidth usage estimated



2.3 Properties of P2P Networks

Chapter 3

Kudzu: An Adaptive, Decentralized File Transfer System

which it has matches (as detailed in Section 3.2.2), the node sends a response back to the node who generated the query. Note that although answering a query may involve opening a new connection, this does

the network without requiring large numbers of connections or excessive query hops through ordinary

chooseNewPeer: Choose and return the next peer in L to which the node is not presently connected. If every peer in the list is presently connected, return *none*. This has the e ect of simply populating the node's available connections with the peers that were initially given to impart specific information about those documents. This will include, for example, common language words that have nothing to do with content (e.g., `a', `the').

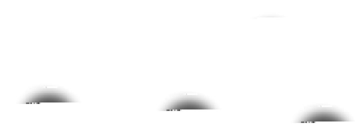


Figure 3.1: A non-optimal separating hyperplane H1 and an optimal separating hyperplane H2 with margin m. Test point T is misclassi ed as black by H1 but correctly classi ed as white by H2.

This is an optimization problem which can be solved computationally using quadratic programming techniques. An example of an optimal separating hyperplane for a binary decision problem in two it should or should not connect to the potential peer. We can thus formulate an organization policy using an SVM classi er as follows:

init

Figure 3.2: A Kudzu network of 5 nodes containing 3 download swarms. Solid lines indicate peer connections, while dotted lines indicate swarm connections.

For each user, the dataset contains two sets of information: the set of queries issued by the user, and the complete set of les shared by the user. Each query consists of a set of keywords and the timestamp at which the query was issued. Each le consists of a lename and a lesize. The dataset only a few minutes or hours rather than multiple months. The choice of time multiple is a tradeo

Chapter 4

Implementation: The Kudzu Client

We have implemented a Kudzu client according to the specification described in Chapter 3, as well as the test harness for running experiments on our client. Since a Kudzu network is comprised entirely of clients with no higher-level coordination required, the client itself implements all aspects of a Kudzu network. Our implementation of the client is a Java program of roughly 3000 lines.

The client is started on the command line and is provided a directory from which to share files and download into and a hostname or IP address of an existing Kudzu peer to connect to. If an existing peer is not provided, the client starts but has no connections, and thus will not be part of any greater network until other peers connect to it. Once the client is started, it presents a simple command-line interface to the network controlled primarily through the following three commands:

```
$ kudzu -d sharedir -n planetlab1.williams.edu
Starting node and connecting to planetlab1.williams.edu...
You are connected to Kudzu.
> query coaster
Sent request for `coaster' to peers.
> responses
Query `coaster':
    id 0: `roller_coaster.mp4' (3907036 bytes):
        Peer planetlab2.williams.edu
        Peer planetlab1.williams.edu
    id 1: `glass_coasters.mp4' (2688476 bytes):
        Peer planetlab3.williams.edu
> downlo4Iams.edu41
```

However, while the RMI implementation was functional, it had several major aws. The most

```
message QueryRequest {
  required string keywords = 1;  // query keyword string
  required bytes requesterAddress = 2; // IP address of requester
  required int32 ttl = 3;  // query's remaining number of allowed hops
}
```

Figure 4.2: One of Kudzu's protocol buffer definitions.

the sequential unsigned integers are used to encode 0, -1, 1, -2, 2, and so forth. This saves bytes when encoding ints whose absolute value is low. Complete protocol buffer messages are also quite

```
message Message {
  required int32 type = 1; // type specifying which (if any) content field is filled
  optional int32 id = 2; // message id to identify a response message
  optional BlockRequest blockRequest = 3;
  optional BlockResponse blockResponse = 4;
  optional ChunkSetRequest chunkSetRequest = 5;
  optional ChunkSetResponse chunkSetResponse = 6;
  optional ErrorResponse errorResponse = 7;
  optional FileStoreResponse fileStoreResponse = 8;
  optional HostRequest hostRequest = 9;
  optional HostResponse hostResponse = 10;
  optional PeerExchangeRequest peerExchangeRequest = 11;
  optional PeerExchangeResponse peerExchangeResponse = 12;
  optional QueryRequest queryRequest = 13;
  optional QueryResponse queryResponse = 14;
}
```

Figure 4.3: Protocol buffer specification of base container message.

4.2 Message Types

Kudzu peers exchange information using 16 distinct message types. We give a brief description of host_request: a request for a random assortment of the target peer's neighbors (not including the requester, of course). The payload contains an int specifying how many new neighbors are desired. This message is used by node organization policies to populate their neighbor sets.

host_response: the response to a host_request message. The payload contains up to the number of requested peer addresses (but may contain fewer).

peer_exchange_request: a request for all known peers in a download swarm. The payload

```
message BlockRequest {
  required sint64 fileChecksum = 1;
  required int64 offset = 2;
}

message BlockResponse {
  required bytes block = 1;
}

message ChunkSetRequest {
  required sint64 fileChecksum = 1;
}

message ChunkSetResponse {
  required bytes chunkSet = 1;
}

message FileStoreResponse {
  required string fileStore = 1;
}

message HostRequest {
  required int32 numHosts = 1;
}

message HostResponse {
  repeated bytes addresses = 1;
}

message ErrorResponse {
  required string errorMessage = 1;
}

message PeerExchangeRequest {
  required sint64 fileChecksum = 1;
  repeated bytes peerAddresses = 2;
}

message PeerExchangeResponse {
  repeated bytes peerAddresses = 1;
}

message QueryRequest {
  required string keywords = 1;
  required bytes requesterAddress = 2;
  required int32 ttl = 3;
  optional int32 id = 4;
}

message QueryResponse {
  required string keywords = 1;
  message FileStubMsg {
    required string name = 1;
    required int64 size = 2;
    required sint64 checksum = 3;
  }
  repeated FileStubMsg matches = 2;
}
```

```
<USER>
    <PROPERTY>
        <USERID>436</USERID>
```

4.3.5 Bootstrapping

Chapter 5

Evaluation

In order to evaluate the e ectiveness of our design and implementation choices, we conducted extensive tests of a Kudzu network using our client by running our test framework on PlanetLab. Of PlanetLab's roughly 1000 machines, we were able to harness roughly half in our tests, which we EvaChaptMetricsaluation

5.1.2 Query Recall

simulation run relative to the amount of traffic actually observed when the dataset was captured. Selecting the most active nodes is an imperfect solution to this problem but serves to compensate

Figure 5.1: Unique query ratios in a network with uncapped TTL.

includes query requests, query responses, and any other messages exchanged on the network. Note that it does *not* include any downloads, since for these tests we did not actually initiate any le transfers when matches were received. Our results are shown in Figure 5.2. A random network organization was used with a minimum connection setting of 3 and a maximum connection setting of 4. These values were chosen to provide a full range of minimal network coverage to near-complete network coverage as the max TTL increased to 10. Furthermore, these values are typical real-world settings { the original Gnutella employed 4 connections per peer.

We see from the curve that bandwidth usage increases signi cantly more than linearly in the TTL; its exponential tendency is particularly pronounced up to TTL 6. More variation is present at higher TTL values, though this likely has to do with the size of the network { with 3 to 4 connections per node, some queries start reaching most of the nodes in the network around TTL 7 and may stop propagating in less than the maximum number of hops. However, the aggregate bandwidth continues to increase steade-1(i)17aloingwithTTL.Thiscoirmisyp(thesis)-btheofTTo io network
tLr,,thatincreating the TTL for queries toer antrhe network is annsicaablep
prob as Though the network bandwidth usage in Our

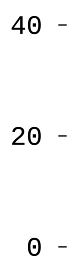
Figure 5.2: Aggregate bandwidth usage across a range of max TTL values.

5.4 Organization Strategies

Given the link between TTL and bandwidth usage, the goal is to maximize query recall while minimizing the TTL (and thus bandwidth usage as well). We investigated the effectiveness of four different organization policies, which we detail here (see Section 3.3.1 for a description of the general policy types). Recall that we refer to the minimum number of peer connections as MIN and the maximum number as MAX. For all of our tests, we set MIN to 3 and MAX to 4.

A fixed policy with random organization. For this organization, the manager assigned each peer in the simulation at least MIN and no more than MAX other peers to connect to. The selection process consisted of randomly picking two peers from the pool of peers with less than MAX assigned connections and pairing them, then repeating until all peers had at least MIN connections or no further pairings were possible. This process was entirely executed on the

bandwidth required in transferring the le stores required to calculate TFIDF values. Recall that a


40
20
0

usual query. However, unlike a query, we impose no TTL on the number of hops and send responses from every recipient node containing a list of the node's current connections. Once all responses have arrived at the initial node, it has enough information to reconstruct the entire network. Note that in the case of the dynamic organization schemes (naive and TFIDF), this will not be an exact

Figure 5.9: Circular network topology resulting from TFIDF organization with passive exploration.

Figure 5.10: Circular network topology resulting from TFIDF organization with active exploration.

Figure 5.13: Download completion CDFs for Kudzu and BitTorrent.

initial seeds would quickly spread to the rest of the swarm (resulting in burst download speeds

5.7 Summary

Chapter 6

Conclusion

6.1 Future Work

While Kudzu is a fully functional P2P le transfer system in its own right, there are some important

6.2 Summary of Contributions

This thesis presented Kudzu, a fully decentralized P2P le transfer system that employs intelligent

Bibliography

[12] Goh, S. T., Kalnis, P., Bakiras, S., and Tan, K.-L. Real datasets for le-sharing peer-to-peer systems. In *DASFAA* (2005), pp. 201{213.

[13] Google Code contributors. Protocol bu er benchmarks. http://code.google.com/p/thrift-protobuf-compare/wiki/Benchmarking, retrieved 20 April 2009.

[25] Ritter, J. Why gnutella can't scale. no, really. http://www.darkridge.com/~jpr5/doc/gnutella.html, retrieved 1 May 2009, February 2001.

[26]