P2P Dictionary for Interactive Groupware Systems

Richard Fung
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
fungr@ucalgary.ca

ABSTRACT

Groupware systems have been popularized by video chat (e.g., Skype), instant messaging, e-mail, and revision management systems (e.g., SVN). Of particular interest to my research are interactive groupware systems, which need to communicate with each other quickly without perceptible delay. In this paper, I design a peer-to-peer (P2P) protocol that shares a dictionary data structure for communication in interactive groupware systems. A P2P protocol allows groupware clients to form an ad-hoc network in private networks and networks isolated from the Internet. Communication in groupware systems can be accomplished using shared memory, and a dictionary partitions shared memory into application-specific chunks. The P2P dictionary is optimized for interactive responsiveness by replicating subsets of the dictionary's entries only to subscribed clients. The design of this P2P dictionary is evaluated in star, ring, line (broken ring), and mesh topologies. I discover that the P2P dictionary scales to a large number of nodes when small data types are stored. The contribution from this research is a P2P dictionary optimized for interactive responsiveness.

1 INTRODUCTION

A groupware system enables collaboration between people using a software system. An ideal groupware system supports real-time communication and collaboration, coordination in a shared workspace, and simultaneous interaction in a workspace [6]. Examples of groupware systems include e-mail and instant messaging systems as well as simultaneous multi-user drawing and document editing applications.

Groupware systems need to communicate with each other using a network protocol. Several communication mechanisms have been investigated in the CSCW (ACM conference on computer-supported cooperative work) community. This work focuses on a shared memory approach, where clients share the same model in a distributed model-view-controller paradigm.[1] Boyle and Greenberg [1] created a rapid prototyping toolkit that uses a shared dictionary to partition shared memory. The toolkit uses a central server to coordinate changes among groupware clients, which I extend by proposing a P2P approach.

In this research, I design a P2P protocol for sharing a dictionary data structure in order to achieve interactive responsiveness for groupware clients. A P2P network scales to many clients without a single bottleneck, but it requires more overhead to handle concurrency compared to a client-

[1] http://grouplab.cpsc.ucalgary.ca/cookbook/index.php/Explainer/HelloWorld

server architecture. A dictionary data structure partitions shared memory into application-specific chunks, which allows clients to subscribe to subsets of the dictionary. Consider the case of a drawing application. A canvas is shared among all clients, but some keys for drawing cursor and colour are client-specific. Only a subset of the dictionary entries needs to be replicated to specific clients.

I consider several design decisions to achieve interactive responsiveness in a P2P dictionary. When a dictionary key is changed, metadata is immediately propagated throughout the network, which makes a best-effort arrival time for each client. Dictionary content is retrieved by delayed propagation: subscribed clients request the actual content only after metadata is updated, which avoids unnecessary transmission of unused dictionary entries.

The ideal network for this P2P dictionary has several dozens of nodes,[2] not exceeding a hundred nodes, in a stable network. Given that the system is designed for interactive groupware, I assume that there is no churn or partitions in the network, which would make the system non-interactive. All nodes are assumed to be trustworthy.

The P2P dictionary is evaluated in different network topologies. A star topology is used to simulate a central server network. Ring and line topologies are used to simulate basic P2P substrates. A mesh topology is constructed from a structured ring substrate by adding a random edge to each node. This paper finds that as the number of nodes increases, a mesh topology performs better in maintaining a low arrival time whereas the star topology becomes a bottleneck. Ring and line topologies, due to their construction, perform worse in almost all experimental conditions.

Any two peers can modify a dictionary entry concurrently using an optimistic approach, which assumes that conflicts rarely happen. If a same-revision conflict happens, the protocol resolves the conflict by a democratic vote (i.e., whoever wins a vote becomes the most recent copy). Frequent-change conflicts are not dealt with in this system.

The remainder of the paper is structured as follows. The paper begins with a background into groupware systems. Then, prior work is reviewed in existing mutable P2P systems. The primary contribution of the paper is presented next, the design and implementation of a P2P dictionary. Lastly, the paper performs an evaluation and concludes the paper.

2 BACKGROUND

Before describing the design of the P2P dictionary, we need a background about groupware systems and dictionary data structures. This section briefly introduces these concepts.

2.1 Groupware System

A groupware system enables collaboration between people using computers. There are several types of groupware systems, which are shown along two dimensions as shown in Table 1. A groupware system can facilitate same-time collaboration when two individuals work together to produce an outcome. A groupware system can facilitate different-time collaboration by saving messages left by another person for retrieval at another time (e.g., e-mail).

[2] I use *node* synonymously for client within the context of a P2P network.

The place where collaboration occurs is another dimension to consider. Same-place collaboration happens in a meeting by sharing a presenter's view on everyone's computer. Different-place collaboration happens when remote participants share the same artifact such as a drawing canvas, but they are geographically remote.

	Same Time	**Different Times**
Same Place	Face-to-face interaction	Asynchronous interaction (geotagged reminders)
Different Places	Synchronous distributed interaction (voice chat, instant messaging)	Asynchronous distributed interaction (e-mail)

Table 1.Groupware time space matrix, borrowed from [5]. Examples are provided in parentheses.

These types of groupware systems have differing requirements. This work focuses on same-time collaboration systems. The requirement for such a system is interactive responsiveness, a best-effort arrival time from a sending to receiving peer. Issues that arise in different-time collaboration such as partitioned networks and dropped peers, therefore, are beyond the scope of this research.

An interactive groupware system imposes specific constraints on the behaviour of an application. A groupware system has some tolerance in its arrival time. A video stream that is 400 ms delayed is annoying, but it does not stop a conversation. The acceptable tolerance varies based on the type of application. On the other hand, a groupware system cannot freeze while data is being requested by a client. For instance, locking a drawing canvas when someone else is updating it is intolerable.

2.2 Dictionary Data Structure

As presented earlier in this paper, a dictionary data structure is used to share memory between clients. Groupware systems may share massive amounts of data, but only subsets of the dictionary are of interest to a given client. A dictionary data structure allows clients to subscribe to specific <key, value> pairs.

To understand how a dictionary structure may be used in a distributed model-view-controller application, consider a drawing application. The drawing canvas must be consistent amongst all peers for collaborative editing (Figure 1). Each client can modify the drawing canvas by adding, modifying, and removing graphical objects. On the other hand, peers do not need to know about another peer's drawing cursor.[3] Thus, each client should only subscribe to its own drawing cursor (i.e., mouse position and drawing colour). Table 2 shows the dictionary entries that could be used to model Figure 1.

The dictionary keys can be named such that global objects are subscribed by all clients. In the example, any key prefixed with "canvas" is required by all clients. Other keys can be prefixed with a client-specific identifier such as "client1" and "client2". A dictionary provides highly flexibility in sharing data to fit the needs of many groupware systems.

[3] Groupware researchers have found advantages in knowing another client's active working area for *awareness*, but it is beyond the scope of this paper.

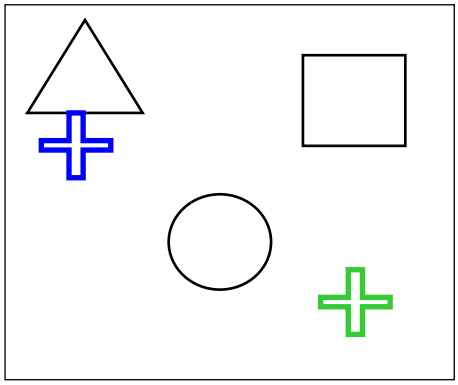

Figure 1. Shared drawing canvas.

		Stored at client		
Key	**Value**	**1**	**2**	**N**
Canvas.Object1		•	•	•
Canvas.Object2		•	•	•
Canvas.Object3		•	•	•
Client1.MousePosition	(20,40)	•		
Client1.DrawingColour	Blue	•		
Client2.MousePosition	(60,90)		•	
Client2.DrawingColour	Green		•	

Table 2. Sample dictionary and corresponding canvas for a groupware drawing application.

3 RELATED WORK

3.1 Dictionary Design

The concept of partitioning shared memory into application-specific chunks of information was previously proposed by Grouplab Shared Dictionary [1] and LIME [16]. Grouplab Shared Dictionary partitions shared memory using a dictionary data structure and API. A central server replicates dictionary entries to its clients, and clients express interest in entries using subscriptions. LIME [16] provides a shared tuple space that stores any type of tuple. LIME does not provide an implementation, only a theoretical design. Both systems have mechanisms to store, retrieve, and listen to updates in shared memory. Interactive responsiveness is achieved in these systems implicitly by pushing data to the clients prior to a client requesting the data.

The main difference between prior work and this work is a detailed network protocol design that does not assume the existence of a central server. This work details the design of a protocol to discover and replicate dictionary entries on a per-client basis.

3.2 Network Design

In order to build a P2P dictionary, I review existing P2P systems that deal with mutable content. This section considers four topics: network topology, data interest, consistency, and caching strategies.

Several network topologies are available for a groupware system. A client-server architecture was taken in mutable systems such as the Grouplab Shared Dictionary [1], but it is not a P2P approach. File systems such as OceanStore [11] and Coda [20] propose a two-tier system where second-tier peers exchange content with each other and consult with first-tier peers for content locations. BitTorrent, Napster [9], and the Eliot file system [22] use a directory server to supply a list of initial peers to contact. Lastly, a network can be completely decentralized with self-organizing clients such as in Gnutella and Freenet [9].

A P2P approach allows data to be distributed over several clients, but there is a choice in how that data is distributed. The Clique replicated file system [17] has interest in all files in a file system. Distributed hash table approaches such as Chord [23] and Pastry [19] assign data based on a hashed value of a key that closely matches a peer's randomly assigned ID. XOROS [4] complements the hash assignment by storing redundant copies at peers closes to the hashed value. Other systems express the data interest based on usage. The Coda file system [20] maintains a per-client cache of most recently used files. BitTorrent uses torrents to specify which files a peer is interested in receiving and sharing. Research in P2P games [2][10] introduce a locality of interest, the area surrounding a player's position in a game map.

After selecting the data of interest, P2P systems must decide if and when to cache content. File sharing systems such as BitTorrent and Gnutella wait until the user issues a download request before pulling them from the P2P network. Multimedia systems such as GridMedia [25] push updated content immediately. Peers can also poll other peers periodically for updates [7][12][18]. A Gnutella extension [12] caches content using a hybrid push-pull protocol by updating immediately after receiving updated metadata and polling neighbours periodically.

Consistency has been a major issue for P2P systems that store mutable content (i.e., content that can be modified by any peer). The Thomas write rule [24] states that the most recent copy should be used. LOCUS [15] uses version vectors that record the number of changes made at each site since a previous reconciliation. The Ivy file system [13] maintains a log of all changes with histories to previous changes. The Clique file system [17] saves version histories, which are checksum logs of previous data [8]. Alternatively, XOROS [4] locks the content before making a change. P2P gaming systems [2][10] employ time-limited locks to ensure game consistency.

The P2P dictionary proposed in this paper extends from the prior work. My P2P dictionary uses a (1) completely decentralized network topology, (2) specifies data of interest using subscriptions (a generalization of user-downloaded torrents), (3) uses a hybrid push-pull protocol for caching data, and (4) ensures consistency based on the Thomas write rule. A fully decentralized network topology is chosen because it forms in ad-hoc networks without requiring any peer to become the "official" node. Subscribing to a minimum subset of keys allows interested peers to maintain data that they are interested in receiving, which is similar to people choosing torrents based on interest. A hybrid push-pull protocol is used because a push protocol achieves the best-effort response time and delayed pull reduces unnecessary network traffic. The Thomas write rule provides a simple way to identify the latest content on the network, using a network's built-in latency as the clock.

4 DESIGN OF A P2P DICTIONARY

The P2P dictionary adheres to the four design decisions iterated in the Related Work section. The P2P dictionary design achieves a subscription-based replication model using the following rule:

> A peer replicates dictionary entries of interest.

This section details the design of a P2P dictionary that is optimized for interactive responsiveness by replication and subscriptions.

4.1 Metadata

The dictionary avoids the lookup cost of a distributed hash table by fully replicating all a dictionary's metadata to each client using a best-effort approach. The metadata consists of tuples with the following entries:

- **Key**: key is used to index a dictionary entry
- **Owner ID**: identifies the client that most recently modified the key
- **Revision**: identifies a global revision counter for the key
- **List of senders**: list of IDs from the Owner ID to a given client
- **Subscribed**: a Boolean value indicating that a key has a subscription

Each client's dictionary is uniquely assigned an ID number. The owner ID is assigned to the client when it writes to a dictionary entry. The owner ID, revision, and list of senders are used together to prevent duplicate messages on the network.

4.1.1 Metadata Replication

Each P2P dictionary notifies changes to the metadata using *immediate propagation*. Immediate propagation describes an approach where updates are *pushed* to a receiving peer from a sending peer; the receiver does not have to request updates.

4.1.2 Duplicate Metadata

To prevent cyclic loops, a peer performs examines the metadata message before forwarding it. A *list of senders* (Table 13) contains intermediate peers, identified by their ID, from the originating peer to receiving peer. If an arriving metadata summary already has the peer's ID in the list, the message is discarded. When forwarding the message, a peer adds its own ID to the list of senders.

A peer determines if a metadata message has been already seen. The *Owner ID* and *revision* uniquely identifies the version of a metadata message. If both values are identical, the message has already arrived from another path and propagated to its neighbours. Thus, the duplicate message can be discarded. The versioning strategy is further explained in §4.2.2.

The first arrival of a metadata update is the fastest update, so the list of senders is saved from the first metadata update. Duplicate metadata updates would have a different list of senders, but they would specify longer paths in the network.

4.2 Data

The dictionary data exists only at subscribed or proxy peers. The additional fields for a subscribed entry are:

- **Value:** actual content
- **Content Type:** Table 6 lists the currently supported content types in the dictionary. A special content type exists to identify removed content.

This section addresses the case where data exists at a subscribed peer. Proxy peers are discussed afterward.

4.2.1 Data Replication

Dictionary data is transmitted by delayed replication: a subscribed peer replicates a dictionary entry after receiving a metadata update from one of its neighbours. An example of this message is presented in Table 14. This avoids content from being sent to an unsubscribed peer, which

conserves network bandwidth and improves the responsiveness for other metadata updates. Transmission delay is assumed to be larger than round-trip latency.[4] The dictionary entry becomes inconsistent when it knows of new content but has not yet received it.

Delayed replication assumes that the sending peer already has the data. If that peer does not have the data, a proxy request is made to another peer, which is described next.

4.2.2 Data Replication via Proxy

If a peer's makes a request to an unsubscribed peer, peers along the list of senders are consulted to retrieve the dictionary entry. Figure 2 (top row) shows an example of a proxy request. Intermediate peers between the originating peer and interested peer become proxy subscribers. Table 15 shows an example message exchange.

The path will lead to at least one peer with the dictionary entry, assuming no peers have departed. If the originating peer departed but another peer has the dictionary entry, a flooded message will get a response for the data. If no other replicas exist in the network, the requesting peer remains in an inconsistent state and an exception is raised in the API.

When creating proxy paths, the design assumes that a peer will request another peer's data indefinitely. Proxy paths, thus, are maintained by subscribing each intermediate node to that dictionary entry. In the case where a proxy path is no longer needed, methods in the API are called to remove the subscription.

An alternative to a proxy path subscription is a shortcut (directly connecting to the peer with content). A shortcut requires that each node's IP address is known and reachable. A popular peer, however, could have too many shortcut connections, thus becoming a star topology and a bottleneck (see the Evaluation section). A proxy path, on the other hand, maintains the same number of in-bound and out-bound edges at each node and distributes the request for popular content onto other nodes (the proxy path nodes).

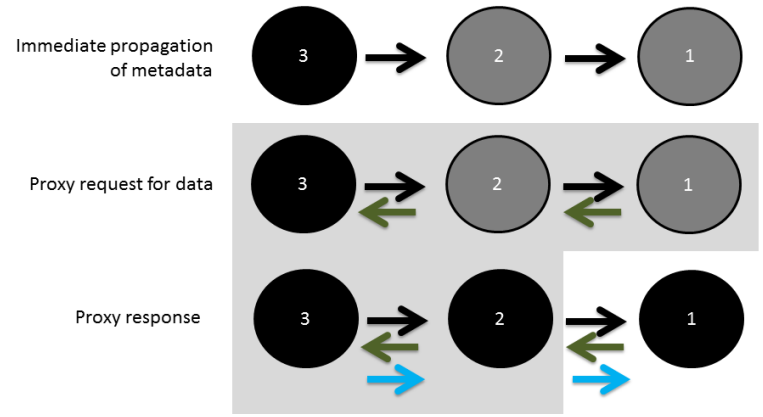

Figure 2. Metadata replication and proxy request. The shaded area highlights a proxy request and response.

[4] The postal mail system has larger round-trip latency than transmission delay. In this case, the sender should send large amounts of data rather than two letters.

4.3 Versioning and Concurrent Changes

Concurrency is handled in the dictionary using an optimistic approach. An optimistic approach assumes that conflicts rarely occur. This P2P dictionary uses versioning with an *owner ID* and *revision number* to handle concurrent changes. *Owner ID* identifies the client who modified a dictionary entry, and *revision* is a global revision number for each dictionary entry.

Peers are consistent when both peers have the same <owner:revision> tuple. An inconsistency occurs when one peer has an older copy of the entry than another peer. The following paragraphs describe these two situations.

4.3.1 Consistent case

Let us begin with peers A and B at <A:0> as shown in Table 3. Both peers A and B want to change a dictionary key *k*. In the normal case, peer A makes the change first and then peer B makes the change after receiving peer A's update. Peer B receives <A:1> and then peer B makes the change <B:2>. Peer B is a direct descendant of peer A's change so peer A can receive the update <B:2> without a problem.

	Version at Peer A	Version at Peer B
Original	<A:0>	<A:0>
Peer A Modifies	**<A:1>**	<A:0>
Peer B -> Peer A	<A:1>	**<A:1>**
Peer B Modifies	<A:1>	**<B:2>**
Peer A -> Peer B	**<B:2>**	<B:2>

Table 3. Consistency is ensured for a dictionary entry using a <owner:revision> tuple.

4.3.2 Conflict case

In a conflicting condition, both peers A and B make a change at the same time before the other peer can receive the update. Two types of conflicts are addressed: same-revision conflicts and frequent-change conflicts.

In a same-revision conflict, both versions are identical but the owners are different. To deal with a same-revision conflict, a democratic vote is conducted. Suppose that peer A has <A:1> and peer B has <B:1>. Both peers increment the revision number by a small randomly chosen integer (implemented as a number chosen randomly from 1–10). Suppose peer A decides <A:6> and peer B decides <B:9>. Neither peer knows about the value chosen by the other peer. After flooding their messages, peer B's revision number is higher and replaces peer A's change. Since any peer in the network can become a proxy subscriber, any peer may participate in resolving a same-revision conflict.

A frequent-change conflict happens when peer A changes a key more frequently than peer B. Peer B does not receive any of peer A's updates, so it is unaware that a conflict existed in the network. The Thomas write rule [24] governs conflicting versions. The most recent transaction is the officiating copy and previous changes are disregarded. The network latency in the network is implicitly used to determine the most recent arrival.

To handle a frequent-change conflict and preserve intermediate changes, a client application should avoid concurrent writes to the same dictionary key. A peer-specific identifier should be included in the key.

4.4 Network Design

4.4.1 Substrate and Construction

The P2P dictionary automatically constructs a mesh topology. The mesh is formed by connecting each node in a ring structure and then adding a random edge. The position within the ring is determined by a randomly chosen 32-bit ID number assigned to each node.

Each peer makes at most two out-degree connections based on the design. Inbound connections from another peer, however, are unbounded to handle the randomly assigned edges. A worst-case scenario, thus, would resemble a star topology where one node is connected to all other nodes.

4.4.2 Joining and Departing Peers

When a new peer joins an existing network, it requests all dictionary metadata from its neighbours. The new peer also updates its neighbours with its own dictionary metadata.

The current design of departing peers is to leave immediately. If the departing peer is the owner of a dictionary entry, that entry will no longer be retrievable. For future work, I intend to add a mechanism to ensure that all requests are serviced before departure and remove unsubscribed dictionary entries.

5 IMPLEMENTATION

The API of the P2P dictionary follows the conventions of Grouplab Shared Dictionary, but the underlying implementation is new for this paper. This section begins by describing the implementation details and then summarizes relevant API details.

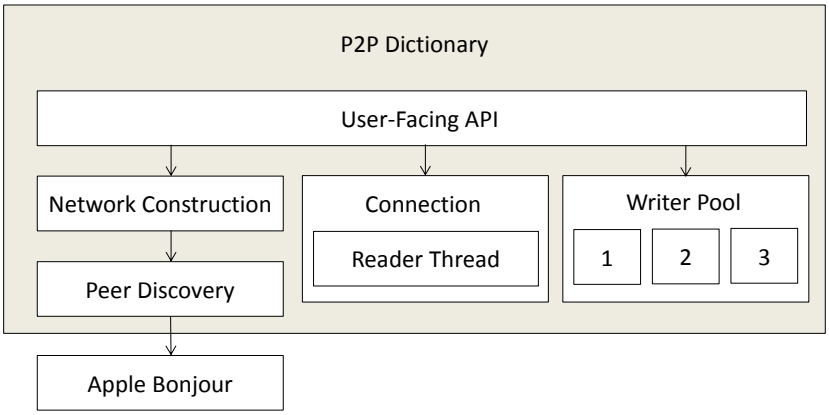

Figure 3. Components of the P2P dictionary.

The P2P dictionary is written in C# using the Microsoft .NET Framework. The P2P dictionary consists of a network construction, connection, and writer component as shown in Figure 3. The code runs independent of third-party libraries except for the network construction component, which uses Apple Bonjour's service discovery protocol. Bonjour requires a local area network for discovering peers. The expected number of nodes in a local area network would be in the hundreds, but it actually depends on the router configuration.

5.1 Network Construction Components

Network construction relies on a third-party library to discover other peers without a central server. Apple Bonjour provides a service discovery protocol that broadcasts over local area networks. The P2P dictionary registers each instance to Bonjour for discovery by other peers.

The peer discovery component maintains a list of registered P2P dictionaries found by Apple Bonjour. Both the node ID and IP address for each dictionary is saved. A P2P dictionary can have more than one IP address if a computer has multiple network cards (e.g., wired and wireless). The node ID prevents duplicate connections to the same node.

The network construction component periodically checks the peer discovery component for new nodes (currently, every 5 seconds). If there are new neighbours, the peer disconnects from the old node and makes a new connection to the new node in order to maintain the same out-degree edges. Since in-degree nodes are formed by random assignment, they are not affected by network construction.

The P2P dictionary can manually connect to any Internet-reachable IP address using API methods. This allows the client software to construct other types of network topologies.

5.2 Connection Component

The connection component uses TCP client-server connections to communicate between peers. The message protocol extends the HTTP 1.1 transport protocol. In HTTP 1.1, a client sends requests to a server and the server responds back. The P2P dictionary relaxes the client-server restriction such that both sender and receiver can request and respond.

The HTTP protocol is extended to allow independent requests and responses. Extra headers as shown in Table 13 are included in the response header to identify the content being delivered.

Requests and response are used to push and pull messages, respectively. Any response may have been propagated through several nodes, which can be inspected through the list of senders in its header. Requests behave identically to normal HTTP and affect a single peer only. HTTP requests can be issued by a web browser to debug P2P dictionary instances.

The HTTP protocol was chosen over alternatives such as XML and binary data because the HTTP protocol is human-readable and asymmetric by design. XML is a symmetric data type, which requires extra fields to delineate a push or pull operation. Binary messages were not chosen because they are difficult to debug.

5.3 Reader and Writer Components

Each TCP connection creates its own reader thread. The reader thread's procedure is:

1. Handle incoming data.
    a. Read the header and contents.
    b. Queue messages to respond to the incoming peer, or otherwise queue messages to forward to other neighbouring peers.
    c. Repeat steps a–b if there is more data on the wire.
2. Sleep for 15 ms.
3. Repeat steps 1–2.

Three global writer threads are maintained for a P2P dictionary. Each writer thread handles one-third of the outgoing messages. A writer thread's procedure is:

1. Empty send queues.
    a. Select one of the peers.
    b. Write *one* message from the send queue.
    c. Select another peer and repeat steps a-b.
    d. Start from the first peer and repeat steps a–c until all queues are empty.
2. Sleep for 15 ms.
3. Repeat steps 1–2.

A known limitation of the reader and writer components is a 15-ms delay in each thread. This delay cannot be further decreased because thread switching in .NET Framework could take 15 ms, a limitation in the runtime environment. The P2P dictionary's behaviour becomes unpredictable if keys are written at a rate faster than 15 ms.

5.4 User-Facing API

5.4.1 Subscription

A client application specifies which keys it is interested in using. The distribution of key subscriptions in a network is specific to the client application, the assignment of a node ID, and the construction of proxy paths. Subscriptions are added using a function call in the API, AddSubscription(pattern) or writing to a dictionary entry with put(key, value). Subscriptions are maintained on a per-client basis.

Wildcard	Meaning
*	Matches any length of string including zero-length strings.
?	Matches any single character
#	Matches any single digit
[charlist]	Matches any character in the charlist
[!charlist]	Matches any character that is not in the charlist
[lower-upper]	Matches any character between lower and upper (using ASCII ordinal rules)

Table 4. Wildcards used for pattern matching, based on the implementation of Visual Basic 6.

5.4.2 API Methods

get(key) retrieves the most-recently received value of a key. The data already exists on the peer before get(key) was called; thus, the method is equivalent to accessing a local dictionary. If the value is not yet received or does not exist, an exception is raised to the caller.

get(key, timeout). To handle a subscribed entry that has not arrived yet, the API provides a method that waits for data, get(key,timeout). The waiting period has a finite timeout (currently, 500 ms). This allows applications with tolerances to interactive responsiveness to wait for data to arrive. After a timeout, an exception is raised to the caller (Table 2).

	Subscribed	Not subscribed
Received	Returns value immediately	Raise an error in the API
Not received	Wait for dictionary entry	Raise an error in the API

Table 5. Consistency for a dictionary entry at a peer.

put(key) stores a value into the dictionary. The peer that calls put(key) becomes the owner of that key in the metadata. A metadata summary of the change is immediately propagated to its neighbours.

5.4.3 Data Types

The Microsoft .NET Framework provides object serialization, which allows many data types to be stored in the dictionary. Basic data types (Table 6) are sent using plain text to avoid the overhead of serialization and improve responsiveness.

Data Type	Size	Signature	Example
Boolean	4 bytes	number/bool	True
Integer	2, 4, 8 bytes	number/int16 number/int32 number/int64	123
Floating point	4, 8 bytes	number/single number/double	123.456
Text	Variable length	text/plain	Hello world
Binary array	Variable length	application/octet-stream	
.NET object	Variable length	application/vs-object	
Removed	0 bytes	application/nothing	
Null data type	0 bytes	application/null	

Table 6. Data types supported in the P2P dictionary. The .NET object data type encapsulates all other serializable data types in the .NET Framework.

6 EVALUATION

The P2P dictionary is designed for interactive responsiveness. This section assesses interactive responsiveness by reporting the arrival times of a key from an originating peer to a receiving peer.

6.1 Setup

The P2P dictionary was evaluated in a simulation environment running on an Intel Core 2 Quad CPU (Q6600) running at 2.4 GHz with 3.25 GB of RAM. The simulation environment was a C# program running on the .NET Framework 4.0. The simulation program spawned several instances of the P2P dictionary in the same process. The network construction and ID number generation were disabled to place nodes in specific network topologies. An extra component was included in the reader and writer threads to simulate network delay. The computer's actual TCP network stack (local loopback) was used to communicate between dictionaries.

The simulation environment measured the arrival time of a dictionary key, which is the time that it takes for an originating peer to send data to a destination peer. This represents a situation where one peer "talks" to another peer. The paired node interests are assigned such that the number of hops between nodes is tested for all conditions. Node i is assigned an ID i, where i is a node from 0 to $n - 1$, and n is the number of nodes. The nodes are selected such that i and $n - i - 1$ subscribe to each other's keys.

The arrival time is measured by an event raised by the dictionary's API. The reported arrival times are in units of simulated time units (*su*). The simulation time for the current set of tests is based on the clock of the running computer; thus, *su* are milliseconds.

Several variables are manipulated to evaluate the P2P dictionary under different conditions:

- **Network topology.** Four network topologies are considered. A star topology is used as a baseline to model a central server model. Unlike a true central server, a star topology fully replicates metadata to all peers. A ring topology is compared, which represents a basic structured P2P network promoted by Chord and Pastry. A line topology is compared to model a performance-degraded ring topology. A mesh network is compared, which is formed from a basic ring topology with an additional randomly-chosen connection.
- **Number of nodes.** The number of nodes ranges between 6 and 50 nodes, which represents a reasonable number of clients in a network.
- **Number of subscribed peers.** A best-case scenario, where each peer subscribes to exactly one other peer, is compared to a worst-case scenario of a full replication. The best-case scenario simulates one peer "talking" to another person. The worst-case scenario simulates everyone talking to everyone else.
- **Message size.** The arrival time is tested under conditions where peers exchange data payloads of 1 KB to 4 MB.
- **Network load.** Between 0% and 100% of the nodes are involved in periodic chatter with another node. This simulates a case where other peers are "talking" with each other.

The responding variables measured are the *arrival times* for dictionary keys between two subscribed peers and the *number of hops* required to receive a dictionary key.

6.2 Results

The results are organized by first omitting the network latency to model a perfect network. Only the queuing delay is considered. Then, the results are compared to a network with delay to simulate geographically-distant peers in a non-optimal scenario.

6.2.1 Queuing Delay and Number of Hops

The first set of experiments assumes an ideal network with no latency. This allows us to examine the effects of queuing delay in various network sizes and topologies.

Nodes	Arrival Time (su)			
	Star	Line	Circle	Mesh
6	109-124	31-390	15-218	31-31
8	109-124	15-592	15-218	15-31
10	93-124	31-780	31-390	15-124
20	78-124	15-1606	15-733	15-265
50	62-124	15-4570	15-1840	15-405

Table 7. Arrival times for four different network topologies compared to the number of nodes in the network. This case has no traffic overhead, no network latency, and paired subscriptions.

	Number of Hops			
Nodes	Star	Line	Circle	Mesh
6	2	1,3,4,5	1,3	1
8	2	1,3,5,7	1,3	1
10	2	1,3,5,7,9	1,3,5	1,2
20	2	1-19	1-9	1-4
50	2	1-49	1-25	1-6

Table 8. Number of hops from a sending to receiving node.

The network topology had a significant impact on the arrival times in the P2P dictionary. The results in Table 7 show that a star topology achieves an arrival time between 78-124 su, regardless of the number of keys or nodes. The reason is that all senders and receivers are only 2 hops in between, thus there are only two queues to deal with (Table 8).

The line and circle topology arrival times are affected by the number of nodes, and thus, the number of hops required (Table 8). Their best-case arrival times outperform the star topology because there is 1 hop from source to destination. The worst-case arrival time is worse than a star topology because the number of hops is linearly proportional to the number of nodes. For example, the worst-case arrival time in a circle topology for 10 nodes (390 su) and 50 nodes (1840 su) is a five-fold increase in both number of hops and arrival time.

An interesting case happens in the mesh topology. Arrival times for the mesh topology are worse than a star topology, but they are on the same order of magnitude. The number of hops grows sub-linearly in the mesh network (see 20 and 50 nodes in Table 8), which gives it a reasonable performance up to 50 nodes.

Both a star topology and mesh topology are suitable for achieving interactive responsiveness in a fast network. A mesh topology can outperform a star topology ~40% of the time in a 10-node network, ~30% of the time in a 20-node network, and ~20% of the time in a 50-node network.

It is interesting to note that existing Chord/Pastry substrates promote a finger table to achieve better-than-linear hops in a ring structure. I have demonstrated that a finger table is not required for network sizes with 50 nodes.

6.2.2 Network Latency

The previous results only considered the queuing delay. Now, I consider the effects of network latency. Network latency is simulated by computing a theoretical bandwidth and transmission delay. Bandwidth delay arises because a physical link transmits data at a limited rate. Transmission delay is the time required to send data over a long distance. The P2P dictionary simulates delay by sleeping the reader and writer threads for the expected latency duration.

The current set of simulations assumes that the network has homogenous 100 Mb/s bandwidth, which is the maximum theoretical bandwidth of a consumer-level router. The transmission latency is chosen from a linear distribution (8–81 ms) using AT&T's network latency measurements,[5] which represents the network latency in the continental United States.

Nodes	Arrival Time (su)			
	Star	Line	Circle	Mesh
4	202-347	25-416	30-59	39-95
6	196-365	52-843	43-455	23-78
8	208-362	21-1401	24-465	18-95
10	218-365	89-1565	42-852	19-449
20	282-578	44-3400	49-1873	23-809
50	645-10220	42-9531	36-4733	25-1296
70	925-20660	75-13536	48-7363	21-1368
100	1412-30885	34-20036	36-10280	27-1723

Table 9. Arrival times for four different network topologies compared to the number of nodes in the network. This case has no traffic overhead, network latency, and paired subscriptions.

Table 9 shows the results with transmission delay. The arrival time degrades significantly for the star topology at 50 nodes because the central node becomes a bottleneck. Both the line and circle topology arrival times grow linearly as the network size grows, which is expected from the

[5] http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html

previous analysis, but they outperform the star topology at 70 nodes because no single peer in the network becomes a bottleneck. The mesh topology shows dramatic improvement: the random edge reduces the number of hops and avoids a bottleneck at any peer. Its arrival times are significantly lower than the other network topologies.

I conclude from this experiment that a P2P dictionary in a mesh topology provides the best scalability at a higher number of nodes. This analysis, however, does not take into account that several peers may share the same network router.

6.2.3 Subscriptions

To test the effect of subscriptions, I compare pair-wise subscriptions with a fully replicated P2P dictionary. The results are presented in Table 10 with only queuing delay and Table 11 with network latency.

Nodes	Arrival Time (su)			
	Star	Line	Circle	Mesh
6	93-124	15-405	31-218	15-31
10	78-124	15-780	31-390	15-93
20	62-109	187-1638	15-717	15-171
50	62-124	15-4258	15-1794	15-280

Table 10. Arrival times for four different network topologies compared to the number of nodes in the network. This case has no traffic overhead, no network latency, and full replication. Compare results with Table 7.

Nodes	Arrival Time (su)			
	Star	Line	Circle	Mesh
6	181-349	66-731	31-377	22-102
10	231-351	35-1647	26-763	31-309
20	489-2523	44-3428	33-1601	51-573
50	829-16228	55-9434	26-4442	46-858

Table 11. Arrival times for four different network topologies compared to the number of nodes in the network. This case has no traffic overhead, network latency, and full replication. Compare results with Table 9.

A star topology's arrival time is mainly affected by the transmission delay and is affected by full replication. A central node takes longer to replicate content to each peer. For the line and circle topology, full replication has negligible effect on arrival times; only one extra request-response loop is added to each client.

An interesting result appears for the mesh topology. In both Table 10 and Table 11, the mesh topology's worst-case arrival time for full replication is better than for pairwise subscriptions, both in queuing and transmission delay (Table 7 and Table 9). The proxy path construction explains this observation. In a minimum-subscription case, only one path from the originating peer to destination peer is used. In full replication, any faster path from the sender to destination can be used.

6.2.4 Message Size

The previous experiments transmitted 18 bytes per dictionary entry (a 64-bit encoded time in ASCII). In this experiment, the dictionary entries vary in size from 1 KB to 4 MB. Table 12 shows the arrival times in a 20-node network with no traffic overhead and pairwise subscriptions.

Theoretically, messages of 12.8 KB should incur a 1 ms delay in a 100 Mb/s link, and the bandwidth delay should eclipse transmission delay at 1037 KB. Experimental results show no conclusive trends messages less than 1024 KB as expected. Experimental results were not conducted at a large message sizes to show insights. A mesh topology, however, is more sensitive to message sizes than a star topology because of an increased number of hops. As an example, a 20-node mesh network with 4-MB messages has the same arrival time as a 100-node mesh network with 18-byte messages.

6.2.5 Network Load

Network load was simulated with adding another set of pairwise subscriptions, which were written to 2 Hz with 1 MB messages. Arrival times were measured for load factors involving 2 to 20 nodes in a 20-node network. The results were similar to Table 9; thus, I could not replicate a situation where other traffic delayed the message of another dictionary entry.

Subscriptions were also tested, which confirmed that a mesh network achieves better arrival times than a star topology in a full replication scenario. For a load involving 6 peers and full replication, the arrival time was 330-2416 su in a star topology versus 38-526 su in a mesh topology.

	Arrival Time (su)			
Size (KB)	**Star**	**Line**	**Circle**	**Mesh**
1	327-551	59-3413	40-1650	40-720
64	311-574	68-3407	36-1748	46-561
128	312-600	36-3538	333-1588	48-680
256	304-578	47-3275	72-1671	24-650
512	300-855	38-3768	31-1979	41-821
1024	335-653	73-3989	44-1920	54-872
2048	286-587	135-4541	61-1977	55-1130
4906	478-694	183-5097	135-2807	118-1757

Table 12. Arrival times as a function of message size. Reported for a 20-node network with no traffic overhead, pairwise subscription, and transmission delay.

6.3 Discussion

If we assume *su* is a direct measurement for milliseconds, then we can compare the results to the acceptable limits for delay. VoIP literature advocates that delays less than 100 ms are imperceptible to a user, delays between 100 ms and 400 ms are tolerable, and delays beyond 400 ms are unacceptable.[6] The results from Table 9 suggest that at most 10 nodes in a star or mesh topology could participate with geographically-distant peers. As the network latency decreases until it is negligible, up to 50 nodes could be supported in a mesh topology (Table 7). The acceptable limits for delay may be more lenient depending on the type of groupware application.

6.4 Limitations

There are several limitations in this evaluation. Several factors affect the results of the arrival times. The arrival times are measured suing the DateTime class, which has a known error of ±16 ms.[7] The simulation is running on a Windows PC with other background processes, which could consume clock cycles while the evaluation was running. Since the process runs on the .NET Framework, garbage collection and just-in-time compilation could occur *at any time* during the experiments. Switching between threads of each P2P dictionary instance incurs delay in the measured arrival time.

There are issues with the accuracy of the simulation models. The simulated network latency and queuing delays does not consider saturation of a shared physical network edge. Also, I assumed that transmission delay follows a linear distribution, which is not realistic for a local area network. The assumption of homogenous bandwidth capacity at all peers might be too optimistic for a heterogeneous network with wired and wireless connections.

[6] ITU recommendation G.114.
[7] http://blogs.msdn.com/b/ericlippert/archive/2010/04/08/precision-and-accuracy-of-datetime.aspx

To counteract the effects of these limitations, I reported all timing results in *su* to indicate to the reader that they are not true millisecond readings. I also made comparisons of two extremes, the best-case network configuration when no latency exists to a poor network configuration where all peers cross geographic boundaries. The actual arrival times would then lie somewhere between the two extremes. I explain the reasoning why the arrival times would vary in different conditions.

7 CONCLUSIONS AND FUTURE WORK

This paper designs a P2P dictionary for interactive responsiveness in groupware systems. A dictionary data structure is chosen to partition shared memory into application-specific chunks. A dictionary's metadata is replicated to all peers by immediate propagation for optimal interactive responsiveness. Subscriptions and delayed replication are used to send relevant dictionary entries to interested clients. An evaluation of the P2P dictionary's interactive responsiveness shows that it scales well with small (sub-kilobyte) dictionary entries.

For future work, I will introduce an adaptive forwarding algorithm where a peer records the subscriptions of its neighbouring peers. Knowing a subscription reduces round-trip latency in sending a metadata notification, receiving a request for data, and then sending the actual data. This addresses the problem identified in the current evaluation where full replication in a random topology performs better than 2-pair subscriptions.

The interactive responsiveness of the P2P dictionary needs to be evaluated in a real network. Delays caused by routers, bandwidth saturation, and wireless signal collusion should be considered in subsequent evaluations. The amount of time required for conflict-resolution mechanism should be determined. Given that this work was inspired by the Grouplab Shared Dictionary, the responsiveness of this dictionary should be compared against a true central server architecture.

Other evaluation metrics should be considered in future work such as bandwidth utilization and per-peer memory consumption. Initial results for memory consumption show the benefits of a mesh topology (Figure 4). Peers should be instrumented to detect duplicate messages, which would determine if network coding is beneficial.

8 REFERENCES

[1] M. Boyle and S. Greenberg. GroupLab Collabrary: A Toolkit for Multimedia Groupware. In *ACM CSCW 2002 Workshop*

[2] N.E. Baughman, B.N.Levine. "Cheat-proof playout for centralized and distributed online games," *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings,* vol.1, pp.104-113 vol.1, 2001

[3] A. Chazapis and N. Koziris, "Storing and locating mutable data in structured P2P overlay networks," in Proceedings of the 10th *Panhellenic Conference on Informatics* (PCI2005), Volos, Greece, November 2005

[4] A. Chazapis and N. Koziris, "XOROS: A mutable Distributed Hash Table," in Proceedings of the 5th *International Workshop on Databases, Information Systems and Peer-to-Peer Computing* (DBISP2P 2007), Vienna, Austria, September 2007

[5] C. A. Ellis and S. J. Gibbs, “Concurrency control in groupware systems,” in *ACM SIGMOD Record*, New York, NY, USA, 1989, vol. 18, p. 399–407.

[6] S. Greenberg and D. Marwood, “Real time groupware as a distributed system: concurrency control and its effect on the interface,” in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, New York, NY, USA, 1994, p. 207–217.

[7] R. Guy, P. L. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication. In *Proceedings of the Workshops on Data Warehousing and Data Mining: Advances in Database Technologies* (ER '98), Springer-Verlag, London, UK, 254-265, 1998.

[8] J. Howard. Reconcile Users’ Guide. Technical Report 98-04a, *Mitsubishi Electric Research Laboratory (MERL)*. Cambridge, MA, June 1998.

[9] K. Kant, R. Iyer, V. Tewari. A Framework for Classifying Peer-to-Peer Technologies, *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on* , pp. 368, 21-24 May 2002

[10] B. Knutsson, H. Lu, W. Xu, B. Hopkins. "Peer-to-peer support for massively multiplayer games," *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies* , vol.1, no., pp. 4 vol. (2866), 7-11 March 2004

[11] J. Kubiatowicz et al., “OceanStore: an architecture for global-scale persistent storage,” ACM SIGARCH Computer Architecture News, vol. 34, p. 190–201, Nov. 2000.

[12] J. Lan, X. Liu, P. Shenoy, K. Ramamritham. Consistency maintenance in peer-to-peer file sharing networks, *Internet Applications. WIAPP 2003. Proceedings. The Third IEEE Workshop on*, pp. 90- 94, 23-24 June 2003

[13] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: a read/write peer-to-peer file system. *SIGOPS Oper. Syst. Rev.* 36, SI (December 2002), 31-44, 2002.

[14] D. Malkhi, L. Novik, and C. Purcell. P2P replica synchronization with vector sets. *SIGOPS Oper. Syst. Rev.* 41, 2 (April 2007), 68-74.

[15] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser, and C. Kline. 1983. Detection of Mutual Inconsistency in Distributed Systems. *IEEE Trans. Softw. Eng.* 9, 3 (May 1983), 240-247.

[16] G. P. Picco, A. L. Murphy, and G. Roman. LIME: Linda meets mobility. In *Proceedings of the 21st international conference on Software engineering* (ICSE '99). ACM, New York, NY, USA, 368-377, 1999.

[17] B. Richard, D. M. Nioclais, D. Chalon. Clique: A Transparent, Peer-to-Peer Replicated File System. *Mobile Data Management: 4th International Conference,* MDM 2003 Melbourne, Australia, January 21-24, pp. 351-355, 2003.

[18] N. Roussopoulos. 1991. An incremental access method for ViewCache: concept, algorithms, and cost analysis. *ACM Trans. Database Syst.* 16, 3 (September 1991), 535-563.

[19] A. I. T. Rowstron and P. Druschel, “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems,” in Proceedings of the IFIP/ACM *International Conference on Distributed Systems Platforms*, 2001, p. 329–350.

[20] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, "Coda: a highly available file system for a distributed workstation environment," Computers, IEEE Transactions on, vol. 39, no. 4, pp. 447-459, 1990.

[21] A. St. John, B. N. Levine. 2005. Supporting P2P gaming when players have heterogeneous resources. In *Proceedings of the international workshop on Network and operating systems support for digital audio and video* (NOSSDAV '05). ACM, New York, NY, USA, 1-6.

[22] C.A. Stein, M.J. Tucker, M.I. Seltzer. Building a reliable mutable file system on peer-to-peer storage, *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on* , vol., no., pp. 324- 329, 2002

[23] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for Inter- net applications. In *Proceedings of ACM SIGCOMM*, San Diego (August 2001)

[24] R. H. Thomas. "A majority consensus approach to concurrency control for multiple copy databases". *ACM Transactions on Database Systems* **4** (2): 180–209, 1979.

[25] L. Zhao, J. Luo, M. Zhang, W. Fu, J. Luo, Y. Zhang, S. Yang. Gridmedia: A Practical Peer-to-Peer Based Live Video Streaming System, *Multimedia Signal Processing*, IEEE 7th Workshop on, pp.1-4, 2005.

9 APPENDIX

HTTP/1.1 200 OK P2P-Dictionary: 12345 E-Tag: "12345.67" Content-Location: /data/key P2P-Sender-List: 12345 Content-Type: application/octet-stream Content-Length: <length of data> Response-To: HEAD

Table 13. Example message used in immediate propagation of a metadata update from a P2P dictionary with ID 12345. The key is owned by this dictionary with a revision count of 67. This is the initiating peer, so the sender list only contains this dictionary's ID.

GET /data/key HTTP/1.1 P2P-Path: 12345 P2P-Dictionary: 6789
HTTP/1.1 200 OK P2P-Dictionary: 12345 E-Tag: "12345.67" Content-Location: /data/key P2P-Sender-List: 12345 Content-Type: application/octet-stream Content-Length: <length of data> Response-To: GET <data contents>

Table 14. Example message used in delayed replication at a P2P dictionary with ID 6789. The request is made to a dictionary whose ID is 12345.

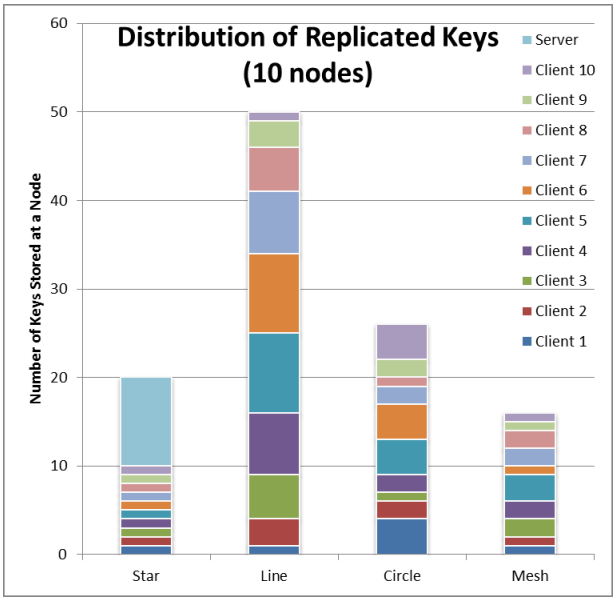

Figure 4. Number of subscribed dictionary entries at each node in a 10-node network.

Peer	Message	Notes
0001 → 0002	GET /data/key HTTP/1.1 P2P-Dictionary: 0001	Peer #1 requests the key from peer #2, which is along the path.
0002 → 0001	HTTP/1.1 305 Use Proxy Content-Location: /data/key P2P-Dictionary: 0002 P2P-Sender-List: 0001 Response-To: GET	Peer #2 does not have the entry. It tells the other peer to make a proxy request.
0001 → 0002	HTTP/1.1 307 Temporary Redirect Content-Location: /data/key P2P-Dictionary: 0001 P2P-Sender-List: 0001 Response-To: GET	Peer #1 makes a proxy request from peer #2. Peer #2 does not have the dictionary entry; thus, it forwards the message to another peer along the path (to prevent loops, it does not to peers in the sender list).
0002 → 0003	HTTP/1.1 307 Temporary Redirect Content-Location: /data/key P2P-Dictionary: 0002 P2P-Sender-List: 0001, 0002 Response-To: GET	Peer #2 makes a proxy request to peer #3 because it does not have the dictionary entry.
0003 → 0002	HTTP/1.1 200 OK P2P-Dictionary: 0003 ETag: “0003.1” Content-Location: /data/key P2P-Sender-List: 0003 Content-Type: … Content-Length: … Response-To: GET P2P-Response-Path: 0001, 0002, 0003 …	Peer #3 has the dictionary entry and returns it to the peers on the proxy path.
	The remaining sequence of push messages is identical to Table 13	

Table 15. Chain of messages used in a proxy request.