

# Characterizing Traffic in Widely-Deployed DHT

Ghulam Memon  
gmemon@cs.uoregon.edu  
Department of Computer & Information Science  
University of Oregon

## 1. INTRODUCTION

A Peer to Peer (P2P) network is one of the most popular forms of distributed resource sharing. The idea behind P2P networks is that several users (peers), distributed globally, share different *resources* among themselves without the presence of a central management authority. Examples of the resources are files, CPU cycles and network bandwidth. The most common use of P2P networks is to share files.

P2P networks are divided into 2 major types: *unstructured* and *structured*. In structured networks, a distributed algorithm ensures that all the peers adhere to a particular structure. This is similar to an IP network, in which all the routers collectively form a tree structure. The presence of a structure ensures that 2 peers can contact each other and share resources by simply following the path dictated by the structure. As a result, if a resource exists in the network then it will be found. In an unstructured network, on the other hand, no such structure exists. Peers join the network by simply contacting random peers. Peers search for resources by flooding the network. As a result, even if a given resource exists in the network, there is no guarantee that it will be found.

The most popular form of structured networks is Distributed Hashables (DHTs). Just like other P2P systems, DHTs are mostly used for filesharing. A DHT imposes a particular structure by assigning unique identifiers to peers, which define their position in the network. The main challenge is how to find location of files in the network. This is achieved by cooperation between peers contributing files (publishers) and peers searching for files (searchers). A DHT peer *publishes* a file by creating a hash of the file. Next, it finds a peer whose ID is closest to the file hash in the ID space. The metric for the closeness ranges from arithmetic difference to pattern matching. Finally, the publisher sends the file meta-data, such as size, name and type to the peer closest to the file ID. This meta-data will be used for future file searches. A DHT peer searches for a file by finding a peer closest to the file hash and retrieving the file meta-data. Finally, by using the file meta-data,

the peer searching for file contacts the file publisher and downloads the file.

The Above discussion indicates that the process of publishing a file as well as searching a file requires finding a closest node to the file ID. DHT peers achieve this by maintaining id, ip and port number of neighboring peers in their *routing tables*. Each peer finds the closest peer to a particular file id by choosing and querying the closest peer from its own routing table. The chosen peer repeats the same process and finds the closest node from its own routing table for the queried file ID and informs the request originator. The request originator then contacts the newly discovered peer. This process continues until the closest peer to the file ID is found.

DHTs have been an active area of research since 2001. The idea of DHTs was introduced by near simultaneous introduction of 4 core DHTs: CAN [1], Chord [2], Pastry [3] and Tapestry [4]. Later studies, such as OpenDHT [15], Accordion [16] and Kademia [7], improved DHT design for real world deployment. However, none of these DHTs have been widely deployed in a real world setting, until recently. As a result, real world properties of DHTs are not well understood. Some earlier studies, such as Performance vs. Cost Framework [17] and Impact of DHT routing [18], analyzed different characteristics of existing DHTs. However, these studies were based on simulations, which cannot account for real world conditions.

A real world DHT can be characterized from 2 different angles, as follows:

- *Usage*: It is important to understand how DHTs are used in real world in order to analyze if the DHT design actually meets user demands
- *Structure*: It is important to evaluate the structure of a real world DHT to see the effect of churn on DHT performance and design.

Characterizing a DHT is challenging because it requires a global view of the network. In the absence of a central component, capturing the global view is difficult. For example, characterizing DHT usage requires the knowledge of exchanged traffic between dif-

ferent peers and characterizing DHT structure requires the knowledge of routing tables of DHT peers. One way to achieve this knowledge is to deploy several instrumented peers in the network at random points. This approach, however, is prone to error. A small number of peers may not be able to collect a representative view of traffic. A large number of peers may disrupt system behavior, thus making the observations unreliable.

In this study, we characterize the usage aspect of a widely deployed DHT, Kad. Kad is based on Kademia. Kad is part of eMule file sharing client [6] with more than 4 million concurrent peers. Our goal is to understand the following:

- How Kad users use the system i.e. *User Behavior*
- How the Kad protocol generates traffic i.e. *Protocol Behavior*
- What are the different characteristics of Kad content

We have developed a technique to capture DHT traffic, without affecting the underlying system. We call this technique *Montra*. We deploy one instrumented Kad peer per regular peer to monitor its traffic. We limit the visibility of monitors to minimize its impact on network. We modify the communication patterns of the monitors such that only the monitored peer is aware of the existence of the monitor. We also address the problem of churn and packet loss.

We implemented Montra in the form of a highly parallel, scalable python based Kad client. Our implementation can monitor 40,000 Kad peers on an Intel Core 2 Duo 2.2 GHz machine with 1 GB RAM while dropping very small fraction of packets. Our software is capable of capturing 50,000 packets per minute at peak loads. We rigorously evaluate Montra using the actual Kad network. We found that Montra is capable of accurately capturing 88%-92% of traffic observed by Kad peers.

Using Montra, we conduct measurements over Kad to characterize the traffic. The analysis of the captured traffic reveals the following important characteristics:

- Majority of popular content loses its popularity within a short period of 2 months.
- As time passes, availability of content decreases.
- 20% of files are searched but never published. 50% of files are published but never searched. 95% of keywords are published but never searched.
- Geographic distribution of peers and content show that peers from all other countries, except China, contribute and consume their fair share of content. Peers from China consume more content than their fair share and contribute lesser content.

- Despite consistent presence of Kad peers from different countries round the clock, traffic rate shows a time of day effect. Further analysis of aggregate traffic reveals that traffic rate is actually driven by European countries

The rest of this report is organized as follows: Section 2 explains the necessary features of Kademia as well as Kad. Section 3 presents Montra in detail, Section 4 describes implementation of Montra, Section 5 describes our validation approach, Section 7 analyzes characteristics of captured traffic, Section 8 reviews the related work and Section 9 concludes the report.

## 2. BACKGROUND

This section provides further background on DHT, with focus on Kademia and Kad.

Scalability of P2P systems comes from the absence of a central component. The downside of such scalable systems is that the clients have to go to extra lengths to locate (search) and contribute (publish) the content. As a result, routing (either structured or unstructured), which is a precursor for search and publish operations, is one of the most important components of a P2P system

Structured Routing approaches came to light with the advent of Distributed Hashtables (DHTs). The usage of a structure implies adhering to a specific geometry, just like IP routing which follows a tree structure. DHTs use such geometries as Ring [2], Hypercube [1] and Tree [3], [4]. DHTs use the geometry to dictate a unique path from one peer to another. It is because of this uniqueness that DHTs can always find a definite answer to a query, as opposed to an unstructured approach. Another integral component of DHTs is the identity of content as well as peers. All well known DHTs proposed up to this point, follow the same approach for this aspect of routing. All DHTs use a non-hierarchical, flat ID space, where each ID comprises varying number of bits (128 bits is the most common number). Both, content as well as peers, are assigned IDs from the same ID space based on a hash function. Content IDs are then associated with the closest Node IDs, based on different notions of closeness such as numeric difference [2], prefix matching [3], [4], XOR distance [7]. DHTs are often used for filesharing systems. Thus the content in that case is files and keywords.

### 2.1 Kademia

Kademia is a DHT, which was introduced in the second round of DHT innovation. As a result, Kademia improves on some of the weaknesses of earlier work and in the process makes DHTs a more practical and usable approach.

Kademia belongs to the class of prefix matching DHTs, just like Pastry and Tapestry. In general, prefix match-

ing naturally maps to a tree structure (e.g. IP addresses). Because of this inherent mapping, Kademia's routing tables collectively form a binary tree which is similar to Pastry and Tapestry. But unlike Pastry and Tapestry, Kademia uses one uniform routing algorithm and distance metric from start to end of the search operation, hence adding simplicity to the design. Kademia uses XOR metric to calculate distance between 2 IDs in ID space, which is simply a form of prefix matching.

The practicality of Kademia comes from its ample use of redundancy. The basic idea is that instead of making one and only one node responsible for a given ID, Kademia allows a given key to be stored at multiple nodes. This proactive approach allows Kademia to handle churn effectively.

Kademia comes with built-in fault tolerance, which is achieved by using *k-buckets*, where *k* is the number of entries in the bucket. Each *k*-bucket corresponds to a single row in the routing table of other DHTs. Unlike earlier prefix matching DHTs, instead of maintaining 1 contact for every prefix in the routing table, Kademia maintains *k* contacts. Increasing *k* increases redundancy in Kademia and hence improves lookup performance at the expense of more bandwidth usage.

Kademia uses parallel lookups to perform routing, i.e., instead of picking 1 next hop contact at each hop, Kademia picks  $\alpha$  contacts. The provision of parallel lookups prevents request originators from a timeout if the next hop peer is not available. The value of  $\alpha$  will vary for different implementations of Kademia. Choosing a high value for  $\alpha$  will result in larger bandwidth consumption and a lower latency. On the other hand, a smaller value of  $\alpha$  will denote a relatively higher latency threshold. In case of Kad (described next), the value of  $\alpha$  is 3.

Kad is a slightly varied implementation of Kademia, which is employed by eMule. In general, prefix matching designs achieve their destination in multiple steps (e.g. IP routing). In the domain of DHTs, this boils down to question of how many bits are improved at each hop. For example, Pastry improves 4 bits per hop and hence requires fewer hops but larger routing tables. Basic Kademia improves 1 bit at each hop whereas Kad improves *b* bits. According to [5], for Kad's eMule implementation  $b=3.25$ . As a result Kad requires larger routing tables. This is the most important difference between Kad and Kademia.

## 2.2 Kad Operation

Kad uses different kinds of messages to perform its regular operation. A Kad client is mostly involved in search and publication of content IDs for keywords and files. It uses a 2 step process for search as well as publish operations. The first step, which we call *lookup phase*, is aimed at finding *n* closest possible alive nodes

to the target content ID, where *n* depends on the type of request. For publish requests  $n = 10$  and for search requests  $n = 300$ . This phase uses REQUEST messages, which carry the target content ID and requested number of contacts *x*. Whenever a peer receives a REQUEST message, it checks its own routing table to find *x* closest contacts to target content ID. The peer then embeds these contacts in a RESPONSE message and sends the message to the peer which sent REQUEST message.

Once *n* alive nodes are found, Kad clients send either SEARCH or PUBLISH messages, depending on the type of request, to these nodes. The structure of a SEARCH message is same for both keyword as well as file search. It simply carries the content ID being searched. The structure of a PUBLISH message, on the other hand, is different for keywords and files. When publishing a keyword, the PUBLISH message carries the content ID of the keyword itself and the content IDs of the files with which that keyword is associated. This message also carries different file attributes, such as, file name, file size and file type. On the other hand, when a file is published, it carries the content ID of file, IP address and port number of file source. The reason that a Kad client stores content at multiple nodes by sending multiple PUBLISH messages is to ensure availability by replication of content. Also, it sends multiple SEARCH messages to retrieve maximum possible results.

In addition to search and publish operations, a Kad client must also maintain its routing table. A client's routing table is maintained by adding newly discovered alive peers and discarding dead peers. In order to check if a newly discovered peer is alive or an existing peer is dead, a Kad client uses heartbeat messages, called HELLO messages. The client sends a HELLO REQ message to check if a peer is alive. If it receives a HELLO RES message then it knows that the peer is alive. Otherwise the peer is considered dead. Both HELLO REQ and HELLO RES messages carry the node ID of the peer that sends the message.

## 3. METHODOLOGY

This section describes Montra in detail. We first highlight the challenges in capturing the traffic observed by DHT peers. Then, we describe our solution to address these problems, in particular, how our approach deals with such challenges as churn, packet loss and avoiding disruption, posed by a large scale real world DHT i.e. Kad.

### 3.1 Challenges in Capturing DHT Traffic

A naive approach for capturing DHT traffic is to deploy a few instrumented Kad clients, placed randomly across Kad network, which passively monitor and log observed traffic. The problem with this approach is

how to determine proper number of instrumented Kad clients. A small number of such clients will not be able to capture enough traffic samples to draw reliable conclusions. On the other hand, a large number of instrumented Kad clients will change the system itself. Thus, the measurement results will be artificial because they will include the effect of the instrumented Kad clients.

Before proposing a solution to this problem, it is important to understand the factors, that affect the traffic observed by individual peers. DHT Traffic can be divided into following two types:

- *Destination Traffic:* A peer receives this type of traffic when it is the closest peer or *destination* for a particular request. The destination traffic depends on a peer’s position. As the position of a peer varies, observed traffic varies as well. This is because the content IDs, for which the peer is destination, vary in popularity.
- *Routing Traffic:* A peer receives this type of traffic when it is an intermediate hop for a request and *routes* the traffic towards destination. The routing traffic depends upon a peer’s visibility. As a peer becomes more visible in the network, more peers use it for routing.

In this study we only focus on destination traffic because of its consistent nature. Destination traffic observed at given network position stays consistent over time but varies across different positions. We will show in Section 7 that most of the traffic characteristics can be derived by observing destination traffic only.

### 3.2 Monitoring a Single Peer

We capture the destination traffic observed by a single peer, called *target peer*  $P_t$ , by placing a *monitor*  $P_m$ , adjacent to  $P_t$  in the ID space such that no other peer lies between  $P_m$  and  $P_t$ . In this scenario, whenever  $P_t$  receives a destination request, it always informs the *request originator*  $P_r$  about the existence of  $P_m$ , even if  $P_t$  itself is the closest peer to the requested ID.  $P_t$  behaves this way because, in Kad, global decisions about finding the closest possible node to a requested ID are made at  $P_r$ .  $P_t$  is only responsible for making a local decision to find  $n$  closest possible peers to the requested ID based on its own routing table. As a result,  $P_t$  always selects  $P_m$ , along-with other peers, for all the destination requests because according to  $P_t$ 's routing table,  $P_m$  is the closest peer to the requested ID. Once  $P_r$  learns about  $P_m$  then  $P_r$  sends the same request to  $P_m$ , which achieves our goal. This process is shown in Figure 1.

Figure 1 shows the interaction between the request originator, the target and the monitor as follows:

1.  $P_r$  sends a request for a given ID to  $P_t$ .

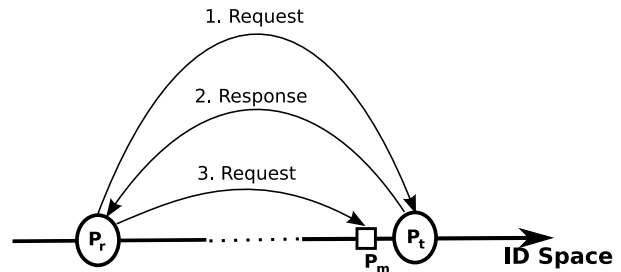


Figure 1: State of ID Space After Addition of Monitors

2.  $P_t$  sends a response to  $P_r$ , which contains the contacts of next hop nodes. If  $P_t$  is the closest node to the requested ID then  $P_t$  will include the ID of  $P_m$  in its response.
3. Since  $P_r$  wants to find  $n$  closest nodes to the requested ID, it sends the same request to  $P_m$  even though the closest node to the request has been found.

This strategy allows us to capture traffic from a single point in a DHT. However, we must observe traffic from several points in the network to capture the global view. We collect large number of samples without affecting peer population by keeping the monitors *minimally visible*. The basic idea is to make the monitor visible to the target node only and not any other node in the network. This enables a monitor to receive all the destination traffic observed by a target node, without affecting any other node in the network. A target node learns about the existence of a monitor when the monitor is added to its routing table. Based on the routing table of a target node, a monitor is always the closest peer for a destination request because the target node does not add itself to its own routing table. Thus, whenever a target node observes a destination request, it will always inform the query originator about the presence of the monitor. As a result, the query originator will send a request to the monitor, which accomplishes our goal. The monitor remains minimally visible by only responding to heart beat (HELLO REQ) messages from the target node. All the other messages from any other node in the network are logged but are not responded to. As a result, every other node in the network considers the monitor as a dead node. But the target node considers the monitor to be alive because it receives responses to heart beat messages.

Such Monitors with Limited Visibility (MLVs) help us in dealing with abnormalities of a large scale real world system, such as Kad, as well. Additional advantages of MLVs are as follows:

- *Avoiding Traffic Overestimation:* MLVs prevent *over estimation* of observed destination traffic. A

request may incorrectly be considered as destination because of Kad's use of UDP packets for communication. More specifically, overestimation of destination traffic can occur in the following scenarios:

- When a request message is lost and not received by the target node. Specifically, when message 1, shown in Figure 1, is lost.
- When a response message is lost and not received by the request originator. Specifically, when message 2, shown in Figure 1, is lost.

In both the above mentioned scenarios, even if the target node is the destination for the request, it will not be considered as the destination because the request originator will consider the target node as dead. An MLV will prevent over estimation of the requests in both the cases because the target node will not be able to inform the request originator about the presence of the monitor. Thus, an MLV will not receive a request in both the cases. On the other hand, a regular monitor, which is as visible as a regular Kad node, would have observed such a request because the information about the presence of the monitor is not restricted to the target node only.

- *Minimizing Disruption:* MLVs minimize system disruption because they do not host any pointers to published content. An MLV must avoid hosting pointers to published content because most of the time, all the monitors will leave the network at the same time, thus resulting in loss of large amount of hosted content. An MLV solves this problem because the monitor is considered dead by all the nodes except the target node, hence it will never receive any content to be published or searched.
- *Coping with Churn:* MLVs solve the problem of churn as well. A single MLV monitors a single target node. As soon as the target node departs the system the associated MLV must stop receiving traffic as well. Otherwise the measurement results will not be reliable. The design of MLVs solves this problem without any additional effort. As soon as a given target node leaves the network, the associated MLV will stop getting all the requests because no other node knows about the existence of the monitor and thus no more traffic can be directed towards this particular MLV.

### 3.3 Identification of Destination Traffic

In this subsection, we address the problem of identification of destination traffic by a given monitor. The problem is how a given monitor can distinguish between

routing traffic and destination and then filter out the routing traffic?

One possible solution to this problem is to filter out the routing requests based on a monitor's routing table. The idea is to modify the monitor code such that whenever it receives a request, it checks its own routing table for a contact which is closer to the content ID. If there exists such a contact, then it considers the request message as a routing message. Otherwise it is considered as a destination message. This solution is faulty because the contact that is considered to be closer by the monitor may have departed the system and the request originator may consider the monitor to be the closest node. It is also possible that the request message from request originator to the next hop peer is lost and thus it may consider that peer is dead and rely on the monitor being the closest peer to the content ID. Thus, in reality, the destination for a content ID entirely depends upon the request originator's view of the interaction.

We solved the problem of distinction between routing and destination traffic, by monitoring a continuous *zone* of ID space. The idea is to monitor all the nodes which share  $n$  high order bits, where  $n$  is the size of the zone. We refer to the  $n$  high order bits as *zone prefix*. For example, if  $n = 8$  and zone prefix = 0xa4 then we monitor all the nodes whose high order 8 bits match 0xa4.

Any request, that matches the zone prefix and enters the monitored zone is bound to have a destination in that zone. As a result, all the requests, which are captured by the monitor, must be destination requests. Some intermediate monitors may observe these requests as routing requests and one monitor finally observes the request as destination request. All the monitors, simply log all the observed requests. When the data collection phase is complete then all the logs are compared. At this point, the problem of distinction between routing traffic and destination traffic is reshaped into finding duplicate requests. The monitor which is closest to a given request is considered to be the actual destination and the requests observed by other monitors are considered as routing requests and discarded. We note that the only restriction on the size of a zone is that of CPU and network resources. We have successfully monitored 6 bit zones on a Core 2 Duo - 2.2 GHz machine with 1 GB RAM, while dropping a very small fraction of packets.

### 3.4 Collecting Additional Information

While the above mentioned approach is extremely lightweight and causes minimal possible disruption, we can only extract the following information from the observed traffic:

- Type of Request: PUBLISH or SEARCH

- Requested Content ID
- Destination Node ID i.e. the ID to which the request was sent

From this data, we cannot distinguish between SEARCH / PUBLISH requests for Keywords and Files. Also, we cannot learn anything about different characteristics of files that are being searched or published. To address these limitations, we extend our technique at the cost of slight disruption to the system.

As mentioned in Section 2, any SEARCH or PUBLISH request from a Kad client progresses in 2 phases. We modified our approach so that when a monitor receives a request from the first phase, it sends a response. The response does not carry any next hop contacts but it informs the request originator that the monitor is alive and can receive a request during the second phase. As a result, our monitors receive SEARCH/PUBLISH requests, which carry additional information about files and keywords being searched/published.

This extension causes a slight disruption to regular operation of the system. As a result of this addition, when content is published, then instead of being published at 10 closest possible nodes, it is published at 9 nodes. Our monitors log the published information but do not maintain it as this will be an additional overhead. Also, when SEARCH requests are sent to 300 closest possible nodes then 299 nodes will respond to the query. Since our monitors do not maintain published information, it is not possible to send responses to SEARCH requests. Also, responding to the request messages results in more visible monitors which leads to system disruption as well.

To avoid these shortcomings of the extension, we respond to requests for a given content ID only once. When we receive an actual SEARCH/PUBLISH request for a given content ID, then we extract the following information about that ID:

- Type of Content represented by the content ID i.e. either FILE or KEYWORD
- Size of the File, provided the content ID represents a FILE.

Once we learn above mentioned information about a content ID then we stop responding to any requests for that ID and treat this information as valid for all further requests.

This extension to our technique is shown in Figure 2, which augments Figure 1 with additional communication. The solid arrow lines show the mandatory messages. If any of these messages is lost then  $P_m$  will not be able to observe the packet. The newly added dotted arrow lines show optional messages that are necessary to extract further information.

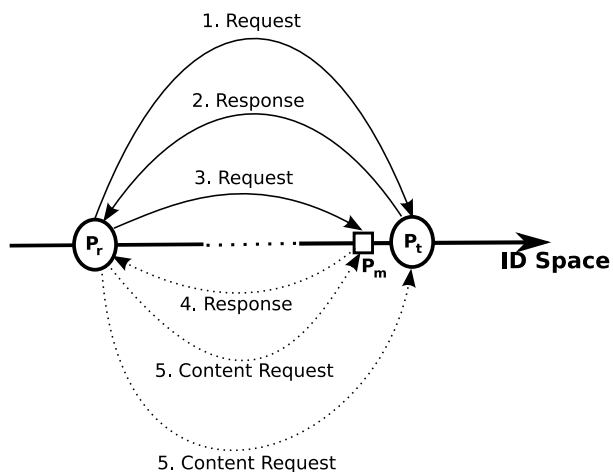


Figure 2: Extended Montra

All the mandatory message exchanges are already described above. Optional message exchanges are as follows:

- 4) If  $P_m$  has never received any request for this ID before then it sends a response to  $P_r$ . The response messages do not contain any contacts. It simply informs  $P_r$  that  $P_m$  is alive
- 5)  $P_r$  sends actual SEARCH/PUBLISH request for content to both  $P_m$  and  $P_t$ .

## 4. IMPLEMENTATION

In order to implement Montra, we developed our own highly parallel, scalable *python* based Kad client, called *Listener*, instead of the one commonly used i.e. eMule. Our approach is lightweight. Also, implementing a customized client helped us to support *Virtual Nodes*. The idea of virtual nodes is that instead of running a separate client to monitor each node, we run only 1 client that simulates the presence of multiple peers by monitoring each node on a different port number. For example, if the client is monitoring a population of 4000 peers then it opens 4000 ports and listens for the messages related to the target nodes. The idea of virtual nodes makes the client more scalable because we don't have to run a separate process per target node. Also, virtual nodes make it possible to maintain a global state that can be shared among monitors. This property is helpful in ensuring that when duplicate requests are received from the same peer for the same content ID at different monitors then only one monitor answers the request, which is the requirement for implementing the enhancement mentioned in Section 3.4

Two important questions that need to be answered in order to implement Montra, are addressed in the following subsections:

## 4.1 Locating all the nodes in a given zone of ID space

In order to find all the peers in a given zone of the ID space, we used the crawler developed by Stutzbach and Rejaie, called *Cruiser* [13]. *Cruiser* starts with a few known contacts in a given zone and downloads their routing tables. In order to download a routing table from a single peer, *cruiser* generates targeted query IDs based on target peer ID. *Cruiser*, then sends request messages for the generated query IDs to the target peer. The peer responds to those queries with response messages, containing contacts, as it would respond to any request message, thus informing *cruiser* about a part of its routing table. By sending such queries, *cruiser* can download the entire routing table of the target peer. *Cruiser* then repeats the same process with every discovered contact in the zone and thus incrementally learns about all the peers in a given zone. *Cruiser* runs on a 6 node cluster.

While crawling, *Cruiser* must avoid peers behind a NAT box. Such peers do not participate in routing and, as a result, do not need to be monitored. These peers are registered with the Kad network because, just like any other peer, they respond to all HELLO REQ messages. However, every message sent by a peer behind a NAT box is sent from a different port, which is open only for a small portion of time. As a result, all routing messages sent towards a peer behind a NAT box are not delivered.

Given the continuously changing peer population of a P2P system, crawling a given zone only once is not adequate. Since our goal is to monitor *all* the nodes in a given zone, we must quickly detect arrival of new peers in the zone. We achieve this by performing back to back crawls i.e. a new crawl starts as soon as the current crawl ends. Back to back crawls also help us to learn about *missed contacts*. As Kad uses UDP for almost all the communication except for file transfer, it is possible that some UDP packets are lost during a crawl and some contacts are left undiscovered.

We note that our approach works accurately without detecting departure of individual peers. When a target node leaves the network, the monitor for that particular node will stop receiving the traffic because of its limited visibility. However, in order to save memory and recycle port numbers, we still detect peer departure by comparing back to back crawls. We discard the departed peers using the heuristic that if a given peer does not respond for  $n$  consecutive crawls then it is considered dead. In our experiments we set  $n = 4$ .

## 4.2 Registering a monitor in a target node's routing table

In order to add a monitor to the routing table of a target node, we use Kad's HELLO REQ and HELLO

RES messages. The HELLO REQ messages are normally used by a Kad client to check if a new or existing peer is alive. A HELLO RES message is sent by peer in response to a HELLO REQ message to confirm its liveness. We use a combination of HELLO REQ and HELLO RES messages, depending on the eMule version used by target nodes, to add our monitors to the routing tables. We add monitors to clients with versions between 0.45a (when eMule started supporting Kad) and 0.47a by sending a single HELLO RES. For versions between 0.47b and 0.48a, we send 2 HELLO REQ messages to each client. This activity is performed by the Listener. The Listener detects the software version by using the information embedded in HELLO REQ and HELLO RES messages and then adapts the sequence of registration messages accordingly.

While implementing the Listener, our goal was to keep it as lightweight as possible, so that it can receive maximum number of packets without dropping them. Thus the Listener's only responsibility is to efficiently send and receive messages. The Listener does not maintain any information about target peers, analyze requests, log requests or generate responses. In order to optimize Listener's performance, we added another component called the *Master*. The Master acts as a coordinator between Listener and *Cruiser*. The Listener establishes a TCP connection with the Master and sends every received packet to the Master. The Master logs all the packets, generates responses based on peer information and sends it back to the Listener, which sends the packet back to the request originator. *Cruiser* also establishes a TCP connection with the Master and informs it about all the newly discovered contacts. The Master maintains this information and informs the Listener to monitor new contacts. The Master also keeps track of those peers which have departed the system. When a peer departs the system, the Master informs the Listener to stop monitoring that particular peer.

The complete setup of Montra is depicted in Figure 3

## 4.3 Implementing the Extension

While implementing the extension mentioned in Section 3.4, our major concern was that how responding to a request message would affect the visibility of a monitor. After carefully examining the eMule code for different versions, we came to following conclusion:

- All the peers with eMule version 0.47a or higher, *do not* add a peer to its routing table when they receive a response to a request message.
- All the peers with eMule version prior to 0.47a add a peer to its routing table when they receive a response to a request message.

eMule versions 0.47a and higher support a different format for request messages as compared to prior ver-

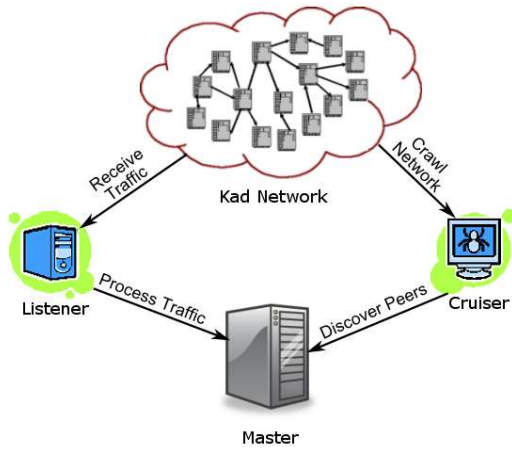


Figure 3: Montra Architecture

sions. Thus, in order to ensure that a monitor remains as invisible as possible, our monitors only respond to the request messages sent by the peers with versions 0.47a and higher. According to our measurements, 70% of peers use eMule version 0.47a or higher.

#### 4.4 Evaluating Implementation

We evaluate Montra implementation by using *packet loss rate* as a metric. Our goal is to find how big a zone our software can monitor without losing significant number of data packets. As the zone prefix length decreases, zone size increases, which results in large number of peers. We evaluate the performance of our software by monitoring zones with different prefix lengths and observing packet loss rate as well as received packet rate. Next, we plot the ratio of packet loss rate to received packet rate against different zone sizes in Figure 4. Data is also shown in Table 1. Figure 4 shows that our software drops most packets when a zone with prefix length 5 is monitored. Figure 4 also shows that our software drops a very small fraction of packets when zones with prefix length less than or equal to 6 are monitored. As a result, we choose 6 bit zones for data collection because we want to monitor the maximum possible number of peers so that we can collect maximum traffic samples. All the datasets used in this study are collected from 6 bit zones.

Zone Prefix Length	Pkt. Loss Rate/Pkt. Recv Rate
5	0.00364
6	0.00009
7	0.00006
8	0.00005

Table 1: Montra Performance

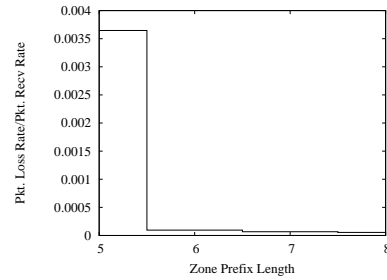


Figure 4: Montra Performance

This section describes our technique to assess the accuracy of Montra. We evaluate accuracy of Montra from two angles:

- *Content Accuracy*: Accuracy with which Montra can observe the destination requests.
- *Peer Accuracy*: Accuracy with which Montra can identify the accurate recipient of traffic.

Our primary concern is the relationship between scale and accuracy. The validations will therefore explore accuracy as a function of zone prefix length. The size of the zone doubles each time the zone prefix decreases by one. The approximate number of peers in each zone is described according to the following formula:

$$\text{peers in zone} = \frac{\text{peers in system}}{2^{\text{prefix length}}}$$

For example, a prefix length of 0 includes the entire system. A prefix length of 1 includes half the system. A prefix length of 2 includes one-quarter of the system, and so forth. At the time of this writing, we estimate the size of Kad at around four million simultaneous peers. Table 2 shows the approximate zone size as a function of the zone prefix length

Prefix Length	Simultaneous Peers
0	4,000,000
1	2,000,000
2	1,000,000
3	500,000
4	250,000
5	125,000
6	62,500
7	31,250
8	15,625

Table 2: Approximate Zone Sizes

## 5. VALIDATION

The simplest way to validate our methodology is to run an eMule based Kad client as a target node, attach

a monitor to this node and then observe what fraction of destination traffic, as seen by the target node, is received by the monitor as well. This strategy, while simple, has following flaws:

- *Overestimation of Accuracy:* When a target node receives a request then there is no reliable way of checking if the request is a destination request. If we naively chose the routing table based approach for determining the status of a request, described in Section 3, then we may incorrectly identify some of the destination traffic, observed at target node, as routing traffic. Such requests are discarded when comparing the traffic observed at the monitor and the target node because we only compare destination requests. This will result in overestimation of accuracy when these requests are not observed at the monitor.
- *Underestimation of Accuracy:* If a UDP response message sent from the target node to the request originator is lost then the monitor will never receive that request. When comparing the destination traffic observed at the target node and the monitor, we may wrongfully underestimate the performance of monitoring technique because the limited visibility of the monitors will prevent such requests from being observed at the monitor.

To avoid these problems, we use the following 2 approaches. These approaches help determine the upper bound and lower bound on accuracy of Montra:

- *Validation using Instrumented Source:* In this approach, we convert a regular Kad client into a random request generator, called Instrumented Source. Instrumented Source targets a given  $n$  bit zone and generates requests for IDs with matching zone prefix and random  $128 - n$  bits. For example, if Instrumented Source is configured to target the 8 bit zone 0xa4 then it will generate random IDs starting with 0xa4. Instrumented Source then logs the node ID which was chosen as destination for the requested ID. While Instrumented Source generates the random requests, Montra monitors the same target zone and logs the traffic. Instrumented Source stops after generating a large number of requests, 2000 in our case. To evaluate Content Accuracy, we check what fraction of issued requests were captured by Montra. To evaluate Peer Accuracy, we check what fraction of captured requests were assigned to proper peers by comparing the destinations observed by Instrumented Source with the destinations observed by Montra.

Introduction of an Instrumented Source solves both the problems of overestimation and underestimation of accuracy, because we know the view of the

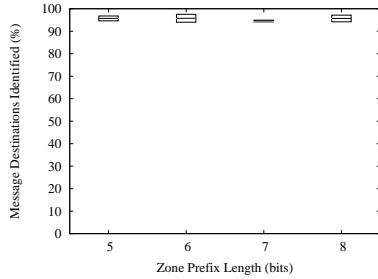
request sender itself. On the other hand, the problem with this approach is that the generated requests may not reflect the actual pattern of system requests or the network conditions seen by a regular peer. As a result, validation using instrumented source gives us an upper bound on the accuracy of Montra.

- *Validation using Instrumented Destination:* In this approach, we convert a regular Kad client into a passive observer, called Instrumented Destination. The Instrumented Destination simply serves as a collector of user generated requests and logs all the observed requests. While Instrumented Destination observes these requests, we monitor its corresponding zone for 6 hours in order to collect sufficient number of request samples. Upon completion of the experiment, we compare the log collected by Instrumented Destination and Montra. To evaluate Content Accuracy, we check what fraction of observed requests were captured by Montra. Evaluation of Peer Accuracy, while limited to the Instrumented Destination only, is complicated because identification of the destination requests at Instrumented Destination is not possible. However, as we are monitoring the entire zone around the Instrumented Destination, we can use the following heuristics to minimize the error in determination of destination requests:

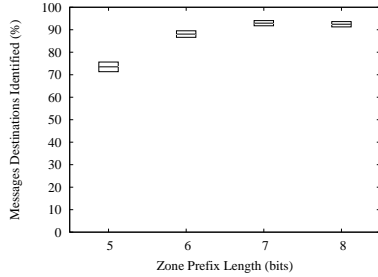
1. For all the requests observed at the Instrumented Destination, if Montra finds a better destination then that request is not considered for evaluating Peer Accuracy.
2. For all the requests observed at the Instrumented Destination, if Montra does not receive the request at all, then that request is not considered for evaluating Peer Accuracy because we are already acknowledging the error while evaluating Content Accuracy.
3. All the other requests are considered for Peer Accuracy.

While validation using instrumented destination does not eliminate the problems caused by accuracy overestimation and underestimation completely, it significantly reduces the resulting error. Thus, this validation technique helps us determine the lower bound on the accuracy of Montra. The biggest advantage of validation using instrumented destination is its use of real world requests for validations, which eliminates the possibility of any artificial inflation of accuracy.

- **Content Accuracy:** Figure 5 presents the Content Accuracy of Montra. Each box reflects the

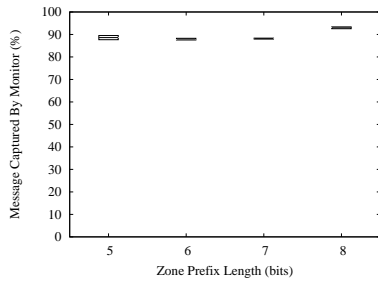


(a) Instrumented Destination Validation

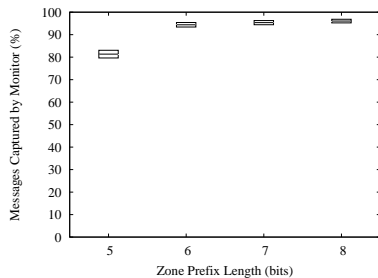


(b) Instrumented Source Validation

**Figure 5: Tool validations; percentage of messages captured by the monitoring tool**

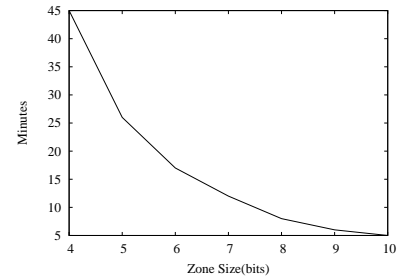


(a) Instrumented Destination Validation



(b) Instrumented Source Validation

**Figure 6: Tool validations; percentage of captured messages where the tool correctly identified the final destination**



**Figure 7: Crawl duration as a function of zone prefix length**

95% confidence interval and the line through the box shows the mean percentage of messages that were successfully captured by Montra. In the instrumented destination experiments, approximately 95% of messages were captured successfully by the monitor regardless of zone size, as seen in Figure 5(a).

In the instrumented source experiments, zone size has an impact, as seen in Figure 5(b). For prefix lengths of 7 and 8, around 93% of messages were captured. For a prefix length of 6, around 87% of messages were captured. For a prefix length of 5, only around 73% of messages were captured.

The reason for this discrepancy is that the instrumented destination experiments artificially eliminate peer churn, since the instrumented destination is always there. The monitor simply attaches to the instrumented destination near the beginning of the experiment. In the instrumented source experiments, the monitoring tool regularly crawls the network looking for newly arrived peers and identifying departed peers. Crawling the zone becomes time consuming as the zone size increases. For example, with a prefix length of 8, the crawl takes around 7 minutes; with a prefix length of 5, the crawl takes around 26 minutes (see Figure 7). Effectively, as the crawl duration increases, the number of peers, which are being monitored, grows.

- **Peer Accuracy:** Figure 6 presents the fraction of monitored messages where the monitor correctly pinpointed the final destination. In most cases, the monitor correctly identified the destination of approximately 90% of the messages. However, in the instrumented source experiment with a prefix length of 5, the monitoring tool correctly pinpointed the destination of only around 80% of the messages. Again, this is attributable to the long crawl duration of such a large zone.

## 6. DESCRIPTION OF DATA

We have collected over 1 TB of data over the past 2 months. Most of data samples are 6 hours long. For some of the analysis we also collected 24 hour long data samples. We collected 6 hour samples by dividing the day into 4 parts: Night, Morning, Afternoon and Evening. We also repeated some of the zones in order to see how the traffic changes over time. Following is a brief description of the data that we have collected.

- **Total Snapshots:** 6 “24 hour” snapshots and 47 “6 hour” snapshots
- **Total Zones Covered:** 44
- **Duplicate Zones:** 5
  - 0x1c (2 snapshots 6 hours each): 03/30/2008 3:00 AM, 04/10/2008 3:00 AM
  - 0x4c (4 snapshots 6 hours each): 03/30/2008 3:00 PM, 03/31/2008 3:00 PM, 04/2/2008 3:00 AM, 04/3/2008 3:00 AM
  - 0x5c (4 snapshots 6 hours each): 03/30/2008 9:00 PM, 03/31/2008 9:00 PM, 04/1/2008 9:00 AM, 04/2/2008 9:00 AM
  - 0x84 (2 snapshots 6 hours each): 03/12/2008 9:00 PM, 04/09/2008 9:00 PM
  - 0xa4 (2 snapshots 24 hours each): 04/12/2008 3:00 AM, 04/13/2008 3:00 PM
- **Non Duplicate Zones (6 hours each):**
  - Night (3:00 AM - 9:00 AM): 10 snapshots
  - Morning (9:00 AM - 3:00 PM): 15 snapshots
  - Afternoon (3:00 PM - 9:00 PM): 11 snapshots
  - Evening (9:00 PM - 3:00 AM): 11 snapshots

## 7. TRAFFIC CHARACTERISTICS

This section describes the important characteristics derived from the captured Kad traffic. Our goal is to understand the traffic properties related to 3 broad categories, as follows:

- Effect of User Behavior on Traffic
- Effect of Client (Kad Protocol) Behavior on Traffic
- Content

We first describe the challenges we faced while deriving traffic characteristics and our solutions to these challenges. Next, we describe different request semantics i.e. what different request types mean when deriving traffic characteristics. After that, we observe the effect of user and client behavior from 2 different angles of Request Target and Request Source. Finally, we describe traffic characteristics based on Content.

For most of this section we use the “Morning” dataset, which we refer to as *Candidate Dataset*. For those distributions of characteristics which need to be compared across different zones, we use *zone envelopes*. The idea is to calculate min and max y values for each x value across all compared distributions and then plot 2 lines denoting the min and max for all zones, thus forming an envelope on zone distributions. The characteristics for which we note that min and max lines are significantly apart, we also mention the Kolmogorov-Smirnov(KS) values, which show the maximum vertical distance between min and max lines. For some characteristics, which cannot be analyzed using 6 hour traffic samples, we use multiple 24 hour traffic samples.

### 7.1 Request Semantics

As mentioned in Section 2, Kad peers send and receive 4 different types of requests i.e. Publish File, Publish Keyword, Search Keyword and Search File. In this subsection we explain how these request types help us in understanding user behavior and client behavior.

- **PUBLISH FILE:** These requests are always sent by the client and thus help us in understanding the client behavior. These requests also signify *availability* of files.
- **PUBLISH KEYWORD:** These requests are always sent by the client and thus play a key role in understanding the client behavior. These request, however, have no effect on availability of content because users are interested in downloading files and not keywords.
- **SEARCH FILE:** These requests are sent by users as well as clients. A user first selects the file to be downloaded, which demonstrates the user behavior. As the download progresses, the client sends more SEARCH FILE requests to find more file sources, which demonstrates client behavior. These requests showcase the *popularity* for a given file.
- **SEARCH KEYWORD:** These requests are always sent by the user and thus demonstrate pure user behavior. These requests also signify the popularity of a particular keyword.

### 7.2 Challenges

This subsection describes the challenges we faced while deriving traffic characteristics.

#### 7.2.1 Effect of NAT

We explore the the effect of NAT because it is likely to affect our ability to separate the user behavior from the client behavior. In order to identify the user behavior, we must identify the users. As Kad uses UDP for

all communication, except for actual file transfer, clients behind a NAT box may send each request from a different port. This makes it impossible to identify a user by its IP, port combination. For example, SEARCH FILE messages are sent by users as well as clients. If the user is behind a NAT box then the client may send repeated SEARCH FILE requests from different ports. If we use IP, port combination as a way to identify users then we may conclude that different users are requesting the same file. But, in reality, we are mixing the user behavior with the client behavior.

PUBLISH requests (both keywords as well as files) are exempt from the anomaly caused by NAT. PUBLISH requests are only sent by peers which are not behind a NAT box. This is because if a peer behind a NAT box publishes a file or keyword then that particular content will not be reachable. As a result when peers behind NAT want to publish some content, they do it through a *buddy*, which is a peer not behind a NAT box. Thus we only examine the effect of NAT for SEARCH requests.

To identify the extent of the NAT effect for SEARCH requests, we examine the following two metrics:

- Ports per IP address: This metric shows the number of different IP port combinations, for a given IP address, from which we received SEARCH requests. This reveals if a single host behind a NAT box sends requests from multiple ports.
- Requests per IP address: This metric shows how many SEARCH requests were received from a single IP address. This is a complimentary sanity check for the first metric. Specifically, we want to observe those instances in which an IP address sent multiple requests and then examine if those requests were sent from different ports.

Figure 8(a) and 8(b) show the CDF for both the metrics for SEARCH KEYWORD requests. Figure 8(b) shows that 18% of IP addresses send 2 or more requests. These are the requests, which we want to examine for NAT effect. Figure 8(a) shows that only 5% of IP addresses, which sent SEARCH KEYWORD requests, were sent from 2 or more ports. Thus the NAT effect appears to be insignificant for SEARCH keyword requests.

Figure 9(a) and 9(b) show CDF for both the metrics for SEARCH FILE requests. Figure 9(b) shows that 45% of IP addresses sent 2 or more requests. This is probably because of repeated requests sent by the clients to discover more file sources. Figure 9(a) shows that in spite of a large number of duplicate requests from a given IP address, only 8% of requests are sent from 2 or more ports. As a result, the NAT effect does not exist for SEARCH FILE requests as well.

## 7.3 Target of Requests

In this subsection we analyze user and client behavior from the perspective of those peers which receive the traffic.

### 7.3.1 Request Rate

Request Rate is the key characteristic that can be driven from DHT Traffic. The rate of requests for a given content ID is calculated by dividing the total requests of a given request type by measurement length, with the exception of SEARCH FILE requests. SEARCH FILE requests show user as well as client behavior, as mentioned in Section 7.1. To separate user behavior from client behavior, we index SEARCH FILE requests by requested file ID and IP, port of the request sender. Using this index, we disregard all duplicate messages. The remaining unique messages give us SEARCH FILE request rate based on pure user behavior. Request Rate helps us to answer following questions:

- *Spatial Properties:* How is the traffic distributed across different IDs for different request types? The answer to this question reveals whether all the IDs (and all the target peers for those IDs) observe same request rate or there exist some *Hot IDs*. We define Hot IDs as those content IDs that observe significantly higher traffic rate than others.
- *Temporal Properties:* How does the request rate for a given ID and request type change over time? The answer to this question helps us to understand how the popularity of a given content ID for a given request type changes over time.

### *Spatial Effect.*

In order to examine spatial effect we consider each request type separately because the request rate for each request type may follow a different distribution. Figure 11(a) - 11(c) show CCDF of request rate for different request types on log-log scale.

Figure 11 shows that different request types have different range of request rates. Figure 11(a) shows that publish keyword request rate has the largest range i.e. some keywords are published at a rate of more than 100 requests per minute. When a file is published, the software automatically publishes all the words in the file-name as keywords. Thus Figure 11(a) shows that some keywords are common in large number of file-names and are published at a much higher rate. Second largest range of request rates is attributed to publish file request rate and is shown in Figure 11(b). Some files are published at 30 requests per minute. Figure 11(c) and Figure 11(d) show that search keyword request rate and search file request rate has the lowest range. Both, publish file and publish keyword requests, have much

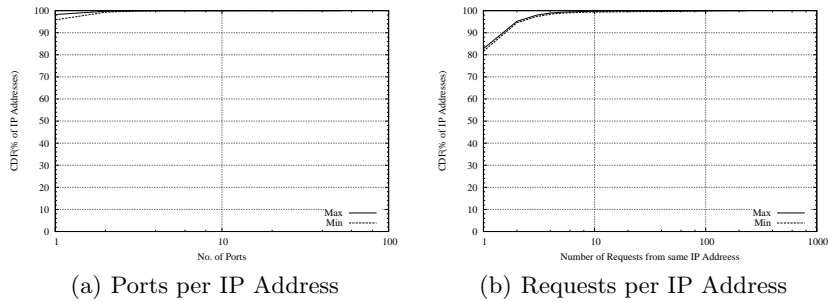


Figure 8: Assessment of NAT Effect on Search Keyword Requests using Candidate Dataset

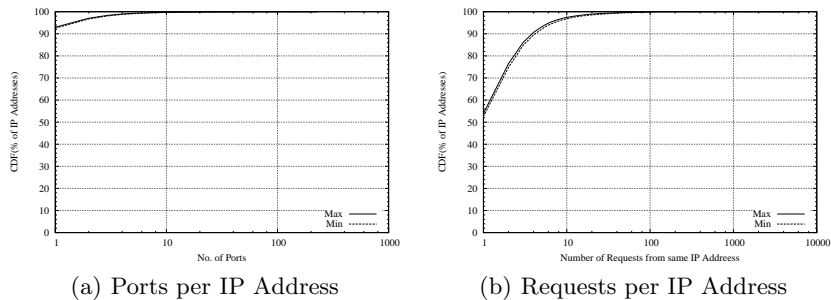


Figure 9: Assessment of NAT Effect on Search File Requests using Candidate Dataset

higher request rate than search keyword and search file requests because publish requests show client behavior and search requests show user behavior. Whenever a user connects to Kad network, the client automatically publishes all shared files and then repeat the process at regular intervals. Search requests, on the other hand, are not automatic.

Figure 11(a), Figure 11(b) and Figure 11(d) show that all the zones in the candidate dataset follow same traffic pattern i.e. majority of content IDs are requested at lower rate while some content IDs are requested at much higher rate. Figure 11(c) shows that traffic rate across different zones vary for search keyword request rate. This is because some zones possess more popular keywords than others.

Figure 11 in general and Figure 11(a) in particular shows that Hot IDs do exist. The largest impact of Hot IDs is observed by the destination peer for a given Hot ID. In addition, peers close to destination peer are also affected by the Hot ID because those peers route the traffic towards the destination peer. We define this impact region surrounding a Hot ID as *radius of Hot ID*. We measure 4 different radii of 4 most popularly published keywords from 4 different zones. We choose popularly published keywords because publish keyword requests are largest source of traffic. We measure the radius of Hot IDs by placing 9 instrumented Kad clients at varying distances from the Hot ID. An instrumented

client is a regular Kad client which is modified to log all incoming request messages. We place the first instrumented client at a distance of 104 bits. A distance of 104 bits means that the instrumented client has 24 high order bits in common with the Hot ID ( $104 + 24 = 128$ , where 128 is the total number of bits used by Kad for peer and content IDs). We pick 104 bits as the initial distance because we have seen from our earlier experiments that a destination peer for any content ID shares 21-24 high order bits with the content ID. We place the remaining 8 clients at increasing distances. Figure 10 shows the radii for 4 measured Hot IDs. The graph legend shows the zone prefix from which the Hot ID was chosen. The x-axis shows the increasing distance of instrumented client from Hot ID. y-axis shows packets/minute observed by instrumented client for Hot ID. Figure 10 shows that most peers observe almost the same request rate as the destination peer upto the distance of 107 bits.

#### Temporal Effect.

We measure temporal effect to understand how the availability and popularity of content changes over time. We evaluate temporal effect by using two 24 hour long snapshots of same zone. The first snapshot was captured on 04/30/2008 and the second snapshot was captured on 06/30/2008, which allows us to study evolution of popularity over 2 month period. Using this data, we

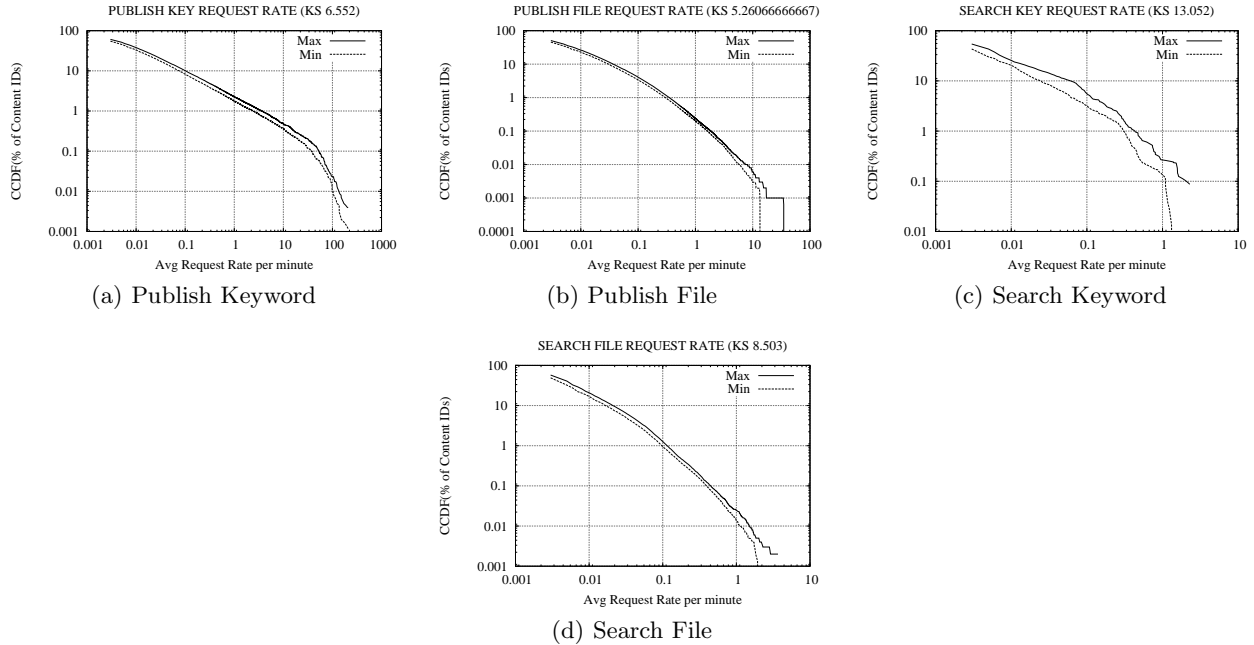


Figure 11: Request Rate based on Candidate Dataset

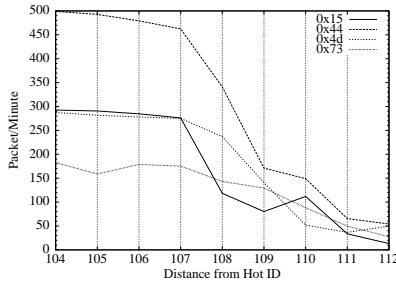


Figure 10: Radius of Hot ID

try to answer following questions:

- **What is the life of most available and most popular content?** We answer this question by comparing top 10 most published and most searched files from 1<sup>st</sup> snapshot against top 10 most published and most searched files from 2<sup>nd</sup> snapshot to see how much common content exists between the 2 snapshots. We also repeat the same measurement for top 100 most published and most searched files from both snapshots. The results are shown in Table 3. Table 3 shows that only 2 of the top 10 most published files from 1<sup>st</sup> snapshot are in top 10 most published files from 2<sup>nd</sup> snapshot and 58 of the top 100 most published files from 1<sup>st</sup> snapshot are in top 100 most published files from 2<sup>nd</sup> snapshot. Intuitively, availability of files should increase with time because as time passes more and more users download the same file. But

the results show that availability of files decreases over time. This is either because the users who downloaded the file deleted the file or they stopped connecting to the network. Table 3 also shows that only 4 of top 10 most searched files from 1<sup>st</sup> snapshot are in top 10 most searched files from 2<sup>nd</sup> snapshot and 44 of the top 100 most searched files from 1<sup>st</sup> snapshot are in top 100 most searched files from 2<sup>nd</sup> snapshot. The results show that in a short period of 2 months, majority of most popular files lost their popularity.

	Top 10	Top 100
Most Published	2 of 10	58 of 100
Most Searched	4 of 10	44 of 100

Table 3: Common Data Between 2 Snapshots

- **How old is most popular content?** We want to understand if the most popular files are recently injected in the system or files gradually become popular over time. To achieve this, we use the same 24 hour snapshots from the last experiment. We check what fraction of newly popular files are genuinely new. We consider those files to be newly popular which exist in top 10 or top 100 most searched files in the 2<sup>nd</sup> snapshot but do not exist in the top 10 or top 100 most searched files from 1<sup>st</sup> snapshot. We call a given newly popular file to be genuinely new if the file is not searched at all in

the 1<sup>st</sup> snapshot. We found that 3 of 6 files (Table 3 shows that 6 of 10 files are newly popular) and 19 of 56 files (Table 3 shows that 56 of 100 files are newly popular) are genuinely new. While these results show that most of the files become popular over time and are not recently injected in the system, Gummadi et. al [12] show that most popular files tend to be recently injected in the system. Gummadi et. al [12] drew this conclusion based on a 6 month long measurement trace. We are in the process of collecting more data to see if our observation continues to hold over a longer period of time.

### 7.3.2 Relation Between Publish and Search Requests

We examine the relationship between PUBLISH and SEARCH requests for files as well as keywords. We explore this relationship because we want to evaluate how evenly supply and demand is matched. If supply is more than demand i.e. more files or keywords are published than searched then clearly resources are being wasted on publishing those files. If demand is more than supply then users of the system will not be satisfied and thus will eventually leave the system. We calculate the relationship using the formula  $P/(S + P)$  for each content ID, where P is No. of Publish Requests and S is No. of Search Requests observed during a measurement window. In order to calibrate availability properly, we identify content contributors by using IP, port combination and count duplicate publish messages from a single client only once. We use the same filter for SEARCH FILE requests. We do not take any such precautions for SEARCH KEYWORD messages because these messages genuinely show demand.

Figure 12(a) and 12(b) show CDF of  $P/(S + P)$  per content ID for files and keywords, respectively. Figure 12(a) shows that 20% of files are searched but not published during our measurement window. Figure 12(a) also shows that 50% of published files are never searched during our measurement window. Figure 12(b) shows 95% of keywords are published but never searched during our measurement window.

## 7.4 Origin of Requests

In this subsection we analyze user and client behavior from the perspective of the peers, which send requests.

### 7.4.1 Geography

We explore geographic origin of request senders in order to answer following questions:

- Which countries around the world are using the Kad network.
- Is the traffic generated by a given country proportional to its peer population. This will help

us understand if users are using their fair share of system resources.

- Do the peers from a given country contribute as much content as they consume?

We achieve this goal by choosing top 5 traffic contributing countries from each zone in the candidate dataset. We then calculate average and standard deviation of percentage of total traffic contributed by each of the top 5 countries. Figure 13(a) - 13(d) show the results for each of the 4 request types respectively i.e. PUBLISH FILE, PUBLISH KEYWORD, SEARCH FILE and SEARCH KEYWORD. The top and bottom lines of the box plot show standard deviation. The middle line shows the mean. Figure 13(a) and 13(b) show that Italy is the biggest contributor of content, i.e. Files as well as Keywords. On the other hand, Figure 13(c) and 13(d) show that China is the biggest consumer of content. This disparity is interesting because the biggest consumers of content should also be biggest contributors of content. This intuition comes from the fact that whenever a client completes downloading a file, it automatically publishes the file back. Thus, this discrepancy demonstrates that either Chinese users manually turn off the automatic upload feature or that a lot of Chinese users are behind NAT boxes and as a result use peers from other countries as buddies.

In order to determine if each of the top 5 traffic generating countries contribute and consume their fair share of content, we use following formula:

$$\frac{\text{Avg. Percentage of Total Traffic Generated by the Country}}{\text{Avg. Percentage of Total Peers Originating from the Country}}$$

Figure 14 shows the results of above mentioned formula for each request type for top 5 traffic generating countries. The horizontal line at 1 shows the perfect ratio of traffic to population. Any requests (either search or publish) above the horizontal line show that peers from a given country are consuming/contributing more content than their fair share. On the other hand, any requests below the horizontal line show that peers from a given country are consuming/contributing lesser content than their fair share. Figure 14 shows that Italy, Spain, France and Brazil publish more content and consume lesser content than their fair share whereas China is the exact opposite.

### 7.4.2 Time of Day Effect

We examine time of day effect to understand if the aggregate rate of traffic changes during different times of day. According to [14], a sizable population of Kad peers from different geographical location is on-line round the clock. As a result, time of day effect on traffic should not exist. However, we show in Figure 15 that such a pattern does exist.

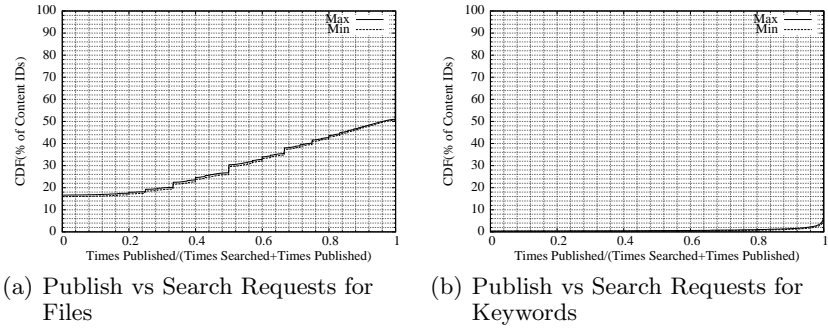


Figure 12: Relationship between Publish and Search Requests based on Candidate Dataset

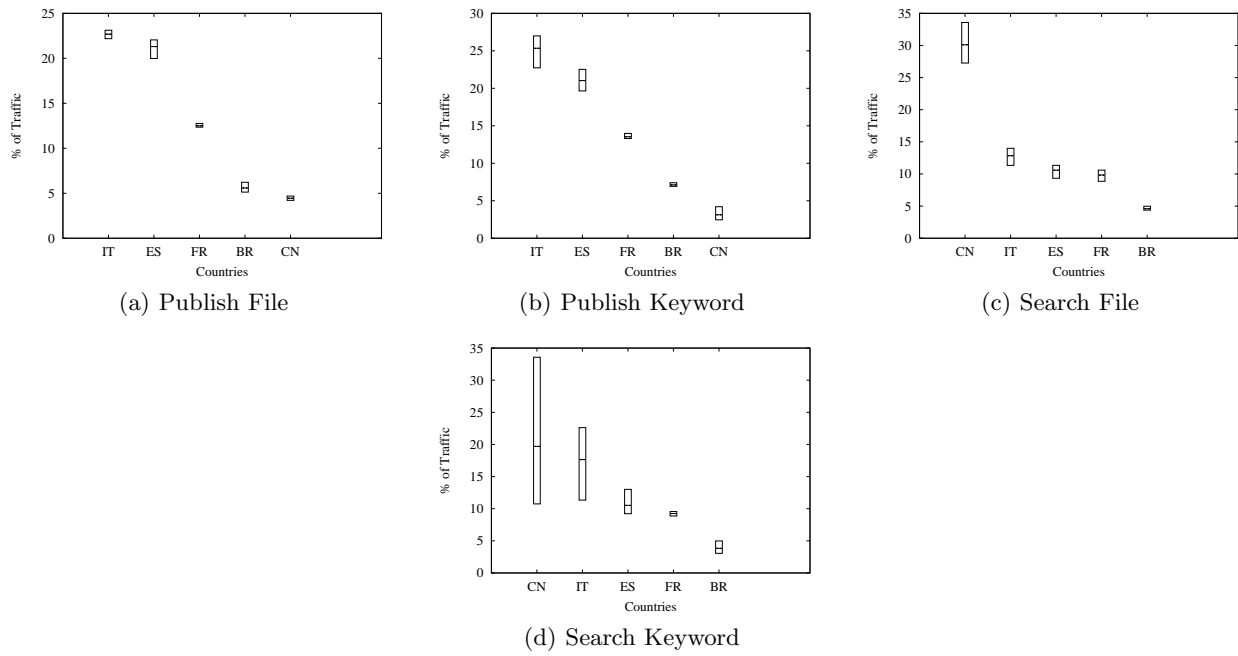


Figure 13: Countries of Origin of Different Requests based on Candidate Dataset

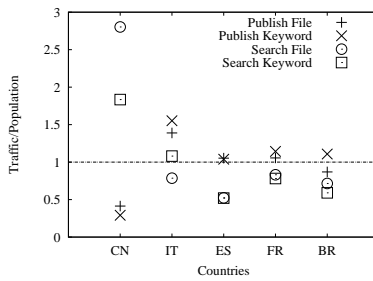
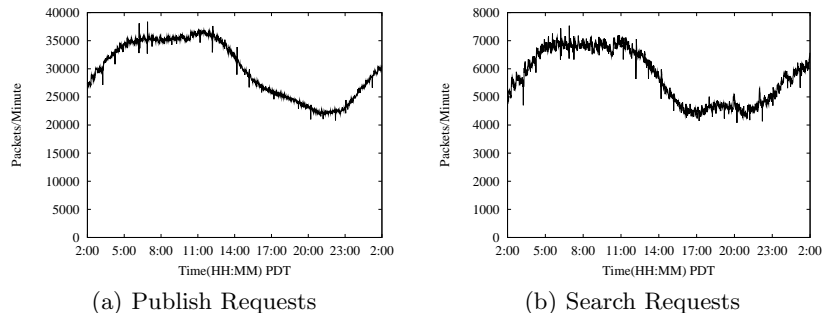


Figure 14: Traffic to Population Ratio



**Figure 15: Effect of Time of Day on Different Request Types using 24 hour logs from zone 0x68**

Figure 15(a) shows aggregate request rate of publish requests for a 6 bit zone, which was observed over 24 hours. x-axis shows pacific time in hours and minutes. y-axis shows number of packets received per minute. Traffic starts decreasing around noon and starts increasing around 11:00 PM.

In order to understand why such a pattern of traffic exists, we look at the publish traffic originating from top 5 countries, namely, China, Italy, France, Spain and Brazil. Figure 16(a) shows that the peak traffic from Italy, France and Spain is observed around noon pacific time, which is 9 PM Rome, Paris and Madrid time respectively. This finding perfectly coincides with [14], which shows that peak population from Rome, Paris and Madrid is observed at 9 PM local time. The figure also shows that peak traffic from China is observed slightly after 6 AM pacific time, which is close to 9:30 PM Shanghai time. This finding is also aligned with results reported in [14]. We conclude from Figure 16(a) that publish traffic is mostly shaped by European countries.

Figure 15(b) shows aggregate request rate of search requests for the same 6 bit zone observed over 24 hours. The shape of the graph, while on a different scale, is similar to Figure 15(a), except that the traffic starts decreasing before noon and there is a slight increase in traffic around 6:00 PM. Also, there is a sharp increase in traffic after 9:00 PM. We know from Figure 13(c) and 13(d) that search requests from China dominate the search traffic. Thus, the similarity between shape of search and publish request rates is surprising.

In order to understand why such a daily pattern of search requests exist, we look at the search traffic originating from top 5 countries, namely, China, Italy, France, Spain and Brazil. Figure 16(b) confirms our earlier finding from Figure 13(c) and 13(d) that China is the leading generator of search requests. However, the aggregate effect of European countries still shapes the aggregate search request rate. We note that the shape of Figure 15(b) is slightly different from Figure 15(a). This is because the decline in aggregate search request

rate is dominated by China.

## 7.5 Content

### 7.5.1 File Size

We examine size of files that are being searched as well as files that are being published. Size of searched files gives us an important insight into user behavior. If the users search (which leads to downloading the file) large files then that means following:

- Kad users are patient: P2P systems usually have no guarantees about file delivery. As a result large files take more time to download.
- Kad users use the system in batch mode: Here, by batch mode we mean that a user issues the command to download the file and then at some point in future the download will be completed. If Kad users are patient then that would imply that Kad users use the system in batch mode.

File Sizes of published files is important to consider so that we can understand if users choose to download files of bigger size because of lack of choice or that they have an option to download files of small size.

Figure 17 shows all the zones included in candidate dataset follow same distribution of file sizes for both types of requests. Figure 17(a) shows that 60% of searched files were larger than 100 MB. This is in contrast with [12], which showed that 90% of files were less than or equal to 10 MB.

Figure 17(b) shows that almost 50% of published files are less than or equal to 10 MB. This shows that while a lot of small files are available, the users prefer large files over small files.

## 8. RELATED WORK

The most closely related work to our approach is Mistral [19]. Our approach is similar to Mistral with regard to the basic mechanism of monitoring the system i.e. a crawler is used to discover the non NAT peers and then

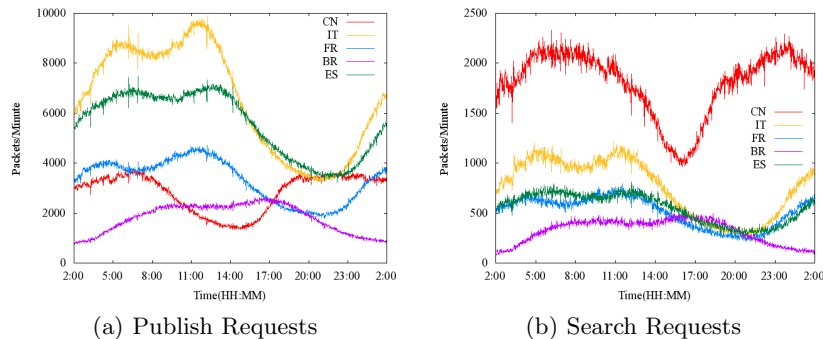


Figure 16: Geographic Distribution of Requests using 24 hour logs from zone 0x68

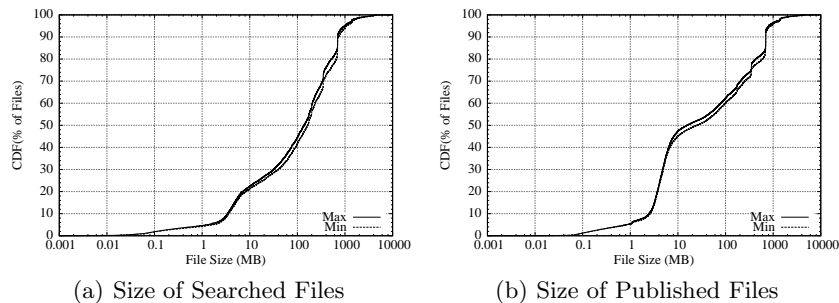


Figure 17: Distribution of File Size based on Candidate Dataset

the protocol features are used to add the monitors to the routing tables of the existing peers. But the focus of our approach is different from Mistral. Our goal is to capture representative traffic samples from the system without disrupting the system. The goal of Mistral is to prove that a DHT can be disrupted by strategic positioning of the malicious peers. The contributions of our methodology, however, are:

- Capture Representative Samples of Traffic.
- Introduction of the notion of distinction between destination and routing traffic.
- Avoid disruption to the system by maintaining absolutely low visibility.
- Gracefully cope with churn.
- Avoid over-estimation of traffic.

The realization that DHT traffic can be divided into destination and routing traffic helps us to group queries for same content ID from same peer into one user request. Thus we can tell, based on user behavior, how frequently a file or keyword is published or searched. Mistral, on the other hand, takes into account all the messages observed and thus the resulting statistics are inflated.

The difference in focus led Mistral to place  $2^{16}$  peers in an 8 bit zone and thus attract a lot more traffic than

necessary. Our approach and validation demonstrates that if placement of monitors is strategically chosen then similar traffic characteristics can be driven with much smaller number of monitors. As a result we are able to monitor 6 bit zones instead of 8 bit zones. Our validation results also show that we can assign requests to final destination peers with sufficient accuracy. This is the result of careful placement monitors in the system. Mistral’s equi-distant placement of peers across the ID space makes it harder to achieve this goal.

## 9. CONCLUSION AND FUTURE WORK

This study presented Montra, a novel technique to capture destination traffic from Kad, without disrupting the system. We discussed the challenges and the solutions for achieving this goal. We implemented our technique in the form of a heavily parallel and scalable python based Kad client. Using this client we validated our approach over the actual Kad network. Finally, we characterized the captured traffic from 3 different angles of User Behavior, Client Behavior and Content.

We plan on collecting more data so that we can derive more statistically sound conclusions about the temporal characteristics of DHT traffic. We are also working on publishing a paper based on this work that will also require making our collected data public. Finally, we plan on developing a P2P traffic simulator based on collected traffic samples.

## 10. REFERENCES

- [1] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, "A Scalable Content-Addressable Network," in *Proceedings of the ACM SIGCOMM*, 2001.
- [2] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *IEEE Transactions on Networking, Volume 11*, 2003.
- [3] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.
- [4] Ben Y. Zhao and John D. Kubiatowicz and Anthony D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location," in *Technical Report: CSD-01-1141*, 2001.
- [5] Daniel Stutzbach, Reza Rejaie, "Improving Lookup Performance over a Widely-Deployed DHT," in *Proceedings of IEEE Infocom*, 2006.
- [6] "<http://www.emule-project.net>"
- [7] Petar Maymounkov and David Mazieres, "Kademlia: A Peer-to-peer Information System Based on the XOR Metric," in *Proceedings of IPTPS*, 2002.
- [8] Moritz Steiner, Ernst W. Biersack, Taoufik Ennajjary, "Actively Monitoring Peers in KAD," in *Proceedings of IPTPS*, 2007.
- [9] B. Krishnamurthy, J. Wang, and Y. Xie, "Early Measurements of a Cluster-based Architecture for P2P Systems," in *Proceedings of Internet Measurement Workshop*, 2001.
- [10] M. Ripeanu and I. Foster, "Mapping the Gnutella Network," in *Proceedings of IEEE Internet Computing*, 2002.
- [11] Subhabrata Sen and Jia Wang "Analyzing peer-to-peer traffic across large networks," in *Second Annual ACM Internet Measurement Workshop*, 2002.
- [12] K. Gummadi and R. Dunn and S. Saroiu and S. Gribble and H. Levy and J. Zahorjan "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.
- [13] D. Stutzbach and R. Rejaie "Capturing Accurate Snapshots of the Gnutella Network," in *Global Internet Symposium*, 2005.
- [14] Moritz Steiner, Taoufik Ennajjary, Ernst W. Biersack, "A Global View of KAD," in *Proceedings of IMC*, 2007.
- [15] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, Harlan Yu, "OpenDHT: A Public DHT Service and Its Uses," in *Proceedings of SIGCOMM*, 2005.
- [16] Jinyang Li, Jeremy Stribling, Robert Morris, and M. Frans Kaashoek, "Bandwidth-efficient Management of DHT Routing Tables ," in *Proceedings of NSDI*, 2005.
- [17] Jinyang Li and Jeremy Stribling and Robert Morris and M. Frans Kaashoek and Thomer M. Gil, "A performance vs. cost framework for evaluating DHT design tradeoffs under churn ," in *Proceedings of Infocom*, 2005.
- [18] K. Gummadi and R. Gummadi and S. Gribble and S. Ratnasamy and S. Shenker and I. Stoica, "The impact of DHT routing geometry on resilience and proximity ," in *Proceedings of SIGCOMM*, 2003.
- [19] M. Steiner, W. Effelsberg, T. En-Najjary and E. Biersack, "Load Reduction in the KAD Peer-to-Peer System ," in *Proceedings of DBISP2P*, 2007