FishNet: File Interchange through a Shared Host Network

Steven Chan, Buddhika Kottahachchi, Hoeteck Wee
{steven,buddhika,hoeteck}@mit.edu

May 10, 2001

Abstract

We argue that while there exist many peer-to-peer file sharing systems incorporating a myriad of features, none of them provide a collection of the most desireable features available in these systems. FishNet fills this void by providing a completely decentralized system which allows for fast resource location, load balancing, file authenticity and efficient storage space management in a single package.

1 Introduction

Peer-to-Peer (P2P) file sharing systems are gaining popularity as a preferred medium for the storage and interchange of digital content. These systems provide a means for users to utilize pooled resources such as bandwidth and storage space made available by other users. For example, an open source developer might wish to publish a software distribution; the average demand for that distribution may be low, but the instantaneous demand after a new release may be too high for the developer's server or network connection to handle. A P2P file sharing system may be better able to handle such an eventuality.

However, we believe that current systems are flawed in varying degrees and fall short of providing users with the true benefits of P2P file sharing networks. For example, systems like Napster [1] use a centralized name resolution mechanism creating a central point of failure. This in turn, compromises data availability and reliability. On the other hand, systems like FreeNet [2] set out to guarantee anonymity and thereby compromise content authenticity.

We propose FishNet (File Interchange through a Shared Host Network), a P2P file sharing system designed to address these concerns of availability, reliability and authenticity. Furthermore, FishNet will better address the generic issues of storage and bandwidth load balancing addressed by most P2P file sharing systems. FishNet operates as a location-independent distributed file system across many different networked computers. FishNet allows authenticated insertion and storage, deletion and retrieval of files. We had six main design goals:

1. Efficient usage of available storage space
2. Effective bandwidth load balancing
3. Mechanisms for authenticating content and preventing unauthorized modification.
4. Decentralization of all network services
5. Robustness & Availability in hostile network environment
6. Consistency of file contents

FishNet is a true P2P file system that incorporates no centralization whatsoever. In spite of this, it supports logarithmic time resource location at any individual FishNet node. An authentication mechanism is provided to enable the system to guarantee publishers that their content cannot be modified by malicious elements. As a consequence of this we also guarantee subscribers that the content they receive is exactly the publisher's intended content.

Additionally, we use a demand based replication system in which popular content will replicate more and live longer on the system. This enables effective load balancing while minimizing content redundancy and improving disk space usage. At the same time, we guarantee that even a file with zero demand will live on the system for at least a week, thereby giving the publisher time to generate demand.

In this paper, we will describe the FishNet system mechanism. Next, we will provide an analysis of FishNet's performance and finally we will provide suggestions on future areas of related research.

2 System Architecture

2.1 Overview

FishNet leverages upon Chord (a distributed hash system discussed in [3]) for name resolution. Each node participating in a FishNet network plays four roles: it acts as a FishNet Client, a File Server, a Location Server and a Chord server all at the same time. Hence, the FishNet software package also consists of these four components.

1. The FishNet Client handles file fetch, publish and delete requests from the user by interacting with the server components of its own node or of other participating nodes. It is also responsible for joining and leaving FishNet networks and other miscellaneous administrative tasks such as the management of File Archives (described later).

2. The File Server component services direct file transfers over TCP/IP to any other node. Of course, a File Server on a certain node can serve up a file only if the node already contains that particular file. We refer to the collection of files that a node can serve as the node's File Archive. Since files may be contained in multiple nodes' Archives, requests for a particular file can be serviced by multiple File Servers.

3. Location Servers hold lookup tables for files and are used to identify which File Servers on the network can service requests for a particular file. Each Location Server is only responsible for a portion of the files on the network while each file on the network is serviced by exactly one Location Server.

4. Finally, Chord Servers identify the Location Server that contains information about a particular file based on its filename. In the FishNet usage of Chord, the keys are FNS filenames while the corresponding value is the list of IP addresses of nodes on the network which contain that file. These key/value pairs are stored on the Location Servers.

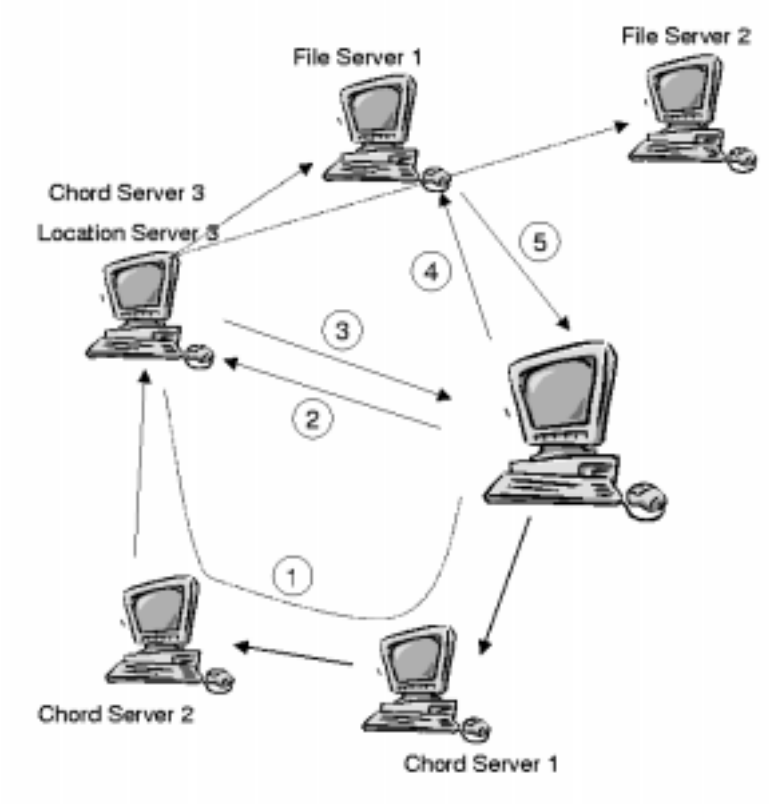
To give an idea of how these four components act together, we will first provide a simplified description of what happens when a file is requested: First of all, the user will provide her FishNet Client with a filename. The FishNet Client looks up this file name using the Chord Servers and is returned with the IP address of the Location Server for that file. A request is then submitted to the Location Server which returns the IP address of one of the File Servers containing the file. Finally, the Client completes the request by approaching the given File Server and setting up the transfer.

Once the Client has completed downloading and verifying the file, the node then becomes another potential File Server for that file and notifies the file's Location Server of this fact. Since every client that downloads a file in turn becomes a File Server for that file, this means that the number of File Servers for a file is directly proportional to the popularity of that file. Hence, the total available bandwidth for downloading each file is also directly proportional to its popularity.

2.2 Details

2.2.1 Joining the network

In order to join FishNet, the user has to know in advance the IP address of another node already connected to FishNet, some of which are listed on the FishNet web-site. The Client program



1. Alice's computer contacts the Chord Server to identify the Location Server.
2. Alice's computer sends file server location query to Location Server.
3. Location Server returns a File Server location.
4. Alice's computer sends file request to File Server.
5. File Server responds with by sending the file back.

Figure 1: The roles played by each node during a download request

contacts the Chord Server at this user-specified IP address and calls `Chord.join` to add the new node to the Chord system. Subsequently it contacts its successor in the Chord ring and copies the key/value pairs for which it has now become responsible. Finally, the node examines its own File Archive and registers its possession of these files with each of their respective Location Servers as described in the next two paragraphs.

For each file in the Archive, the Client extracts the filename and calls `Chord.lookup(filename)` to determine the Location Server for that file. Subsequently it calls `LocationServer.addHost(filename, file date and time, node's IP address)` on the Location Server to add itself to the list of File Servers hosting the file.

If the Location Server returns `TRUE`, this means that the node has been successfully added to the Location Server's list of File Servers for that file. Otherwise, a `FALSE` reply would indicate that the file has already been removed from FishNet (either because the publisher has updated the file with a newer version or because the publisher has removed the file from FishNet entirely) and hence should also be removed from the node's File Archive.

2.2.2 Publishing a file

When a user wishes to publish a file, her Client first determines which Location Server will host the lookup table for the file by calling `Chord.lookup(filename)`. The Client then contacts the Location Server and calls `LocationServer.addFile(filename, file date and time, publisher's IP address)`.

On receiving this request, the Location Server will check if its lookup tables already contain

an entry for a file with that name. If that filename is not already contained in the lookup table, the Location Server regards this as a new file insertion. The Location Server first authenticates the publisher (described in more detail in sections 2.2.9 and 2.2.10) then adds that filename to the lookup table with the publisher's IP address as the only entry in the list of File Servers that contain the file. It also takes note of the file date and time for versioning purposes before returning True to the Client.

If the filename is already found in the Location Server's lookup table, the Location Server then checks the file date and time recorded in the File Archive. If the file being added is as old as or is older than the file pointed to in the Archive, the Location Server merely ignores this addFile request and returns False to the client. The rationale behind this is that a request to add an old file may have been a stale message delayed in transit. Alternatively, this request could also be from a user publishing a file again because the original publication acknowledgment message from the Location Server had gotten lost.

On the other hand, if the filename is already contained in the lookup table but has a newer date and time, the Location Server then regards this as a request to update the copy of the file. After the publisher is authenticated, the Location Server then contacts all File Servers of the older version and sends them an instruction to delete their older copy. Subsequently the Location Server deletes the old table entry and then adds a new entry as if it were a new file insertion using the method described above.

2.2.3 Deleting a file

Deletion of a file is simple. When the Client receives a file deletion request from the user, it first finds the Location Server using `Chord.lookup(filename)`, then submits a request for the Location Server to delete the file. After successfully authenticating the user, the Location Server then sends a message to all the current File Servers of the file requesting they delete it from their File Archive. Finally, the entry in the lookup table is deleted.

2.2.4 Requesting a file

Although the mechanics of a file request were discussed briefly in the overview, we will reiterate them here while filling in the details that were left out initially.

Upon receiving a file request from the user, the Client determines the file's Location Server using `Chord.lookup(filename)` then submits a `LocationServer.getFile(filename)` request to the Location Server. When this file request is received by the Location Server, the Location Server then checks its lookup table for the file entry. If this entry is not found, it returns a `file not found` error. Otherwise, it selects a File Server from the list of hosts for the file and returns its IP address to the Client. The selection of File Servers from the list is done in a round-robin fashion to provide a mechanism for load-balancing.

Following this, the Client attempts to contact the given File Server to retrieve the file. If the file download completes successfully, the file is first checked for authenticity (discussed in Section 2.2.10) before it is saved to the File Archive and copied to a directory of the User's choice. Finally, the Client contacts the Location Server and sends a `LocationServer.addHost(filename, file date and time, IP address)` message to the Location Server requesting that the Location Server add itself to the list of File Servers hosting the file.

On the other hand, if the file download failed somehow (possibly because the transfer was termi-

nated prematurely, or the File Server went offline), the Client then sends a `LocationServer.badHost (filename, File Server's IP Address)` message to the Location Server and the Location Server then responds with the IP address of an alternate File Server for the Client to try.

Note that if a Location Server receives three consecutive `LocationServer.badHost` messages for the same file, it will then delete this File Server from the list of hosts for this file. This is based on the assumption that the File Server must have gone offline or may have lost the contents of its File Archive.

2.2.5 Leaving the network

The protocol for leaving FishNet is intentionally kept simple so that the network operations would be minimally disrupted even if a node on the network were to disconnect without warning. When a FishNet node leaves the network, the Client examines the File Archive and contacts the respective Location Servers calling `deleteHost(filename, node's IP address)` for each of he files in the Archive. Its departure is signaled by the Chord protocol to the node's successor. The successor then becomes responsible for the keys that the leaving node originally held.

2.2.6 Optimizations for increased robustness

Even though FishNet as described thus far would be fully functional, further optimizations need to be put in place to provide for greater robustness of the system.

One such optimization is that each Location Server stores a mirror of its predecessor's keys in a secondary (inactive) table. At the same time, each node sends pings to its predecessor every ten seconds. In this way, even if a node were to be disconnected from the network without leaving gracefully, its successor would detect this by a series of ping timeouts and automatically assume responsibility for the keys that were mirrored by moving the entries from its secondary table to its primary table. This mirroring is continued by copying the new predecessor's keys over to the now empty secondary table.

In the actual FishNet system, we design it such that each key is mirrored in not just one but two successive nodes. In this way, the functionality of FishNet will be unhindered even if two adjacent nodes on the Chord ring were to simultaneously disconnect without leaving gracefully.

Another optimization that we made was that each node on FishNet would constantly republish its files at a user-defined interval of at least once a week. The reason for doing this is to provide a mechanism for recovery in the unlikely situation that a file were to disappear from the network. In this way, even if all the Location Servers on FishNet were to spontaneously lose their lookup tables, files would disappear from the network only for a short period of time (as defined by the user).

2.2.7 Network protocols used

The Chord API has its own means of communication with Chord Servers. On the other hand, the methods of communication between Clients, File Servers and Location Servers presents us with another design choice. In FishNet, we have decided to use the HTTP/GET [4] method for all interactions between Clients, File Servers and Location Servers.

This choice of network protocol provides us with various benefits. Firstly, HTTP is a tried and tested protocol with reasonable performance. Secondly, it is widely understood and thus easily

implemented. This allows for the quick independent development of different FishNet software packages compatible with multiple platforms. Finally, HTTP seems to be very well suited for file transfer – the key element of FishNet. For example HTTP has built-in functionality for resuming interrupted downloads. The streaming capabilities of HTTP would also make FishNet a convenient tool for the distribution of streaming media.

2.2.8 Storage and Replication

Content on FishNet is stored on a collection of Archives. Each node has its own Archive which is locked such that it can only be accessed by the FishNet package. This Archive contains the original version of files downloaded over FishNet, and another user accessible copy is saved. This is done to prevent users from modifying the files stored on their own node and thereby compromising the validity of the content being served by the system.

We require that the user allows the Archive size to always be greater than

```
Sum of the size of files downloaded in the past week over FishNet
+ Size of file about to be downloaded over FishNet
```

The FishNet client will warn the user when this requirement is not met and provide the user with the opportunity to fulfill it. If the user chooses not to allow FishNet to use more space, the pending download will not be allowed to proceed. Files in the Archive are replaced using an LRU (Least Recently Used) mechanism. This means that if the Archive meets the above space requirement but still lacks space to store the pending download, FishNet will make space by removing files starting with the last file served by this node.

Therefore, the Archive size constraint is useful to guarantee that a file will live on that node for at least a week. Furthermore, it eliminates the ability for users to download files without sharing their own resources.

While our storage mechanism helps us achieve the design goal of guaranteeing content authenticity, another of our design goals is to make better use of the available disk space. The problems we observe in other systems with regards to this extend from the replication mechanisms they use to enable load balancing. For example, systems like FreeNet engage in highly redundant replication that does not necessarily reflect future demand patterns.

We observe that predicting future demand is no easy task and cannot be addressed effectively using current technology. Instead our attempt is to minimize unnecessary replication. To achieve this, files stored on FishNet are only replicated based on user demand.

Replication on FishNet is achieved by using the fact that a file in demand is downloaded by many different users. Since all FishNet users are themselves nodes on the network, we achieve replication simply by making those new copies also available for download through FishNet. This eliminates any excessive file replication, but yet provides sufficient replication to meet growing demand. Plus, once demand dies down the LRU mechanism described above will perform passive garbage collection. Therefore, replication on FishNet at any given time is entirely demand based, and provides for the efficient use of storage resources.

2.2.9 Naming Scheme

A FishNet publisher is identified by her public key and her E-Mail address, that constitutes her FishNet username:

```
struct {Publisher's Public Key              |
        Publisher's E-Mail Address}
```

The public key provides a secure means of indentifying the user, whereas the E-Mail address is included only to resolve an unlikely conflict of public keys.

The name of a file in the FishNet Naming Scheme (FNS) has the following structure:

```
struct {Publisher's FishNet username        |
        Arbitrary name of publisher's choice}
```

For instance, this document may be identified on FishNet as:

```
fJ4MycSI_3KVFcq-QxfYG|fishnet@mit.edu|FishNet Paper
```

The publishers public key, is part of a PGP RSA public/private key pair which the publisher generates and uses to sign the files he creates. Since the generated public key cannot necessarily be typed in through the standard keyboard we store it as a sequence of `{A..Z, a..z, 0..9, -, _}` characters. This gives us 64 unique characters, which in turn can be used to encode 6 bits each. Therefore, we could store a 128-bit public key as a sequence of 22 of these characters (last 4 bits are dropped).

The second portion of an FNS file name is the publisher's E-mail address. We use the E-mail address because it provides a unique identifier which does not necessarily compromise anonymity. In addition, this eliminates namespace conflicts whereby two different publishers put up files with the same name.

Finally, we have an arbitrary name selected by the publisher. We impose no restrictions on this part of the name, which allows a great deal of flexibility for the publisher. For example, if desired he can create his own hierarchical naming system somewhat like that used in UNIX. Such a naming system would be beneficial for a publisher who posts content of different kinds (ie. music, software, graphics etc.). However, on the other hand if the publisher only publishes a single kind of content he can choose to exclude the hierarchical file name.

2.2.10 Security

In order to ensure that the retrieved document is the same as the published one, we sign each document by appending a MAC (message authentication code) to the document, that is computed by encrypting a message digest of the document contents. In FishNet, we use the MD5 algorithm [5] to compute the message digest, and then encrypt that with the RSA private key of the user.

$$sign(M) = M + RSA_{priv_A}(MD5(M)) + pub_A$$

To verify the authenticity and the integrity of a message, we could compute the MD5 digest of the message, encrypt the signature of the message with the RSA public key of the user, and check that they are the same. This ensures that malicious users cannot publish documents with another user's FishNet username, nor may they tamper with the contents of the documents without being detected.

To prevent unauthorized modification of a document, we verify the identity of the publisher before publishing a modification as follows. Suppose the client C on Alice's machine attempts to update a file F on the location server L (L is identified via Chord):

1. C sends a file modification request for file F to L.
2. L generates a random string R, encrypt R with the public key on the file F, and send the encrypted string $E(R)$ to C.
3. C verifies that the challenge comes from L by checking the IP address, decrypts the challenge with her private key to obtain R', and sends R' back to L.
4. L checks that R' matches R, and if so, allows the modification to take place.

This ensures that a user requesting to make a modification is the same as the one who published the original document.

3 Analysis

3.1 Discussion

1. *Need for frequent updates.* If a publisher disconnects herself from the network, we only guarantee that her file exists on the system for at least one week. In the case where there are frequent and regular requests for the file, it will persist on the network as long as the demand remains that way. For files that are seldom requested, the publisher may need to re-publish the file regularly, at least once a week. While our FishNet client is specified to republish files at least once a week in an attempt to deal with this potential problem, this republication may not be a satisfactory solution for users who demand stronger persistency of files.
2. *No bandwidth estimation.* The location server chooses a host for file downloads in a round-robin fashion, regardless of the bandwidth of the network connection to the host and the round-trip time between the requesting and the destination hosts. One way to optimize downloads is therefore to have each user specify the bandwidth of their network connections in the software (as with popular streaming media clients such as RealPlayer). The location server can then vary the frequencies with which a host is chosen for downloads based on the bandwidth of its network connection and when the host was last chosen, so that a host with a higher bandwidth and that was last recently used is most likely to be chosen.
3. *Lack of source authenticity.* FishNet does not verify the actual identify of a user when she sets up a FishNet username; for instance, anyone may publish documents with a FishNet username like `fJ4MycSI_3KVFcq-QxfYG|gates@microsoft.com`, as long as she knows the corresponding private key. It is up to FishNet community to establish a web of trust relating users' true identities to their FishNet usernames.
4. *No file segmentation.* There are two disadvantages to the decision to not break a file into smaller blocks. First, some number of nodes that requested for a particular file must have sufficient storage capacity to store the file in its archives to provide the required replication. Second, the load of serving a large document is not distributed, and this could hose the network of a File Server that serves some 10 MB file for instance.
5. *Vulnerability to DoS attacks.* A malicious user could attempt to limit access to specific file by identifying the location servers for the file and launching a simultaneous denial of service attacks on these servers via a flood ping or a smurf attack.

3.2 Performance

Let us suppose there are N hosts and N publishers on the network, and that each publisher publishes an average of k files, so that there are kN distinct files on the network. In addition, let B denote the average bandwidth amongst the FishNet nodes, h the size of each the queries made to the location servers and the subsequent Chord queries, and l the delay due to latency and processing overhead for each connection.

1. *Load Balancing.* Neglecting garbage collection, the number of hosts that serve a particular document is given by one plus the number of hosts that had previously requested for and successfully downloaded the document (one coming from the publisher himself). This allows us to effectively distribute an instantaneous 100% increase in the demand for a file across all of the hosts that had previously downloaded the file (assuming they are all still up), thereby minimizing the delay till the start of downloads. The download time will then depend only on the size of the files, the bandwidth of the connection, and the time to contact the location and file servers. Thus, for a file of size S, the download time is given by $(h/B+l)\log N + S/B$.

2. *Failure Handling.* Suppose a random 5% of the nodes in the network crash, with a reasonably long period of time between successive failure, the system will still work perfectly and lose almost no data (except for those files that no one requested for). Location server data remains intact as the delay will allow the "buddy server" to back up the list of hosts on the machine that went down, and the files will in general not be affected as the files will continue to be replicated across the remaining servers as long as there is a consistent demand for the file.

   On the other hand, if a random 5% of the nodes in the network crash simultaneously, there will be some data lossage, though popular files will be affected only minimally. One problem with such a simultaneous crash is that it is quite likely that both the location server and its backup for a particular file may be gone down simultaneously. In this case, as long as the original publisher for the file remains online, a new location server will be set up for the file, and downloads may resume after that. In the event that all of the hosts that store the file have crashed, then the file is basically lost for good, unless the publisher updates the file again. This will however only happen with very unpopular files, and we assert that the loss is minimal in this case. For files that are replicated across more than 5% of the nodes, we are guaranteed that some node still has the file. Even for files that are replicated across only 1% of the nodes, it is extremely unlikely that all of the nodes that host this particular file are amongst the nodes that crashed. In particular, the probability is given by $\binom{0.05N}{0.01N}/\binom{N}{0.01N}$.

3. *Scalability.* Our system scales effectively with the number of nodes. The location servers store the IP addresses of the individual hosts, which is not affected by the number of nodes on the network, or changes in the nodes. By building our naming scheme on top of the Chord system, we ensure the mapping of files to location servers scale effectively with the number of nodes and to changes in the node configuration.

   Note that an increase in the number of nodes in two ways: first, a long amount of time will be needed to identify the location server via the Chord system. On the other hand, we get improved reliability as the files are replicated over a larger number of computers.

4. *Data Consistency.* Our system is able maintain data consistency almost all of the time. When a file is modified, all the entries in the original location servers are purged, and a new location server is designated for the modified file. As such, old references to the file will report a "File Not Found" error. Suppose a malicious user attempts to claim that a file has been modified

and circulates a new URL for the file. The remaining users could then check whether the old URL still remains valid, and if so, the new URL must be incorrect. This is because the old entries only get purged if the publisher of the new file could correctly respond to a challenge that comprises an encryption based on the public key of the old file.

4 Conclusion

FishNet enables a community of users to efficiently share moderate-sized files. Since FishNet is a true P2P system, it brings with it none of the problems associated with centralized systems. FishNet uses co-operating nodes spread over many computers in conjunction with the Chord interface to provide logarithmic time resource location.

FishNet ensures high availability of popular content by using a demand based replication system. This also provides efficient usage of the storage space available by eliminating extraneous replication. As a side effect, this also enables effective load balancing. Additionally, FishNet also incorporates security mechanisms which guarantee content authenticity. Thus, it incorporates all the desired functionality as stated in our design goals.

Since FishNet is intended to be used by a community, we enforce the requirement that users participating in the network be willing return to the community by serving files to other users. Firstly, this prevents users from leeching off the network. Secondly, this ensures that FishNet is kept running with optimal efficiency.

While we believe that FishNet is a well designed system, this by no means suggests that there remains no room for improvement. Examples of areas that could do with more work are in the areas of bandwidth estimation and in the initial determination of other FishNet nodes for joining the network. Even without these improvements, we are convinced that FishNet has the potential to become the de facto standard for file sharing across the Internet.

References

[1] Napster http://www.napster.com/

[2] the free network project http://freenet.sourceforge.net/

[3] Dabek F., Brunskill E., Kaashoek M. F., Karger D., Morris R,, Stoica I., and Balakrishnan H., Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. In the Proceedings of the *8th Workshop on Hot Topics in Operating Systems (HotOS-VIII),* Schloss Elmau, Germany, May 2001.

[4] Fielding R., Gettys J., Mogul J., Frystyk H., and Berners-Lee T., RFC 2068 - Hypertext Transfer Protocol – HTTP/1.1, 1997.

[5] Rivest R., RFC 1321 - The MD5 Message-Digest Algorithm, 1992.