Updating Adaptive, Decentralized Peer to Peer Networks

Gordon Worley
redbird@rbisland.cx

Abstract

Peer to peer networks make it possible for individuals to share files without the need of servers. While such networks offer many benefits and are excellent at file sharing, they are currently useless for many other tasks, such as hosting a collection of hyperlinked documents. Adaptive, decentralized peer to peer networks like Freenet are capable of changing this, but such networks lack a means of updating files. This paper outlines a proposal for updating such a network. It discusses the security issues surrounding updating and deals effectively with them. Also, it looks at the possible consequences of updatable peer to peer networks and why and where it would be most beneficial.

Introduction

How can an adaptive, distributed peer to peer network be updated? Before this question can be answered, some background on past means of updating information on networks, particularly the Internet, is needed. In the beginning, information was (and still is) stored in a central location and served to clients by having them connect to the server and download the desired information. Updating such a client-server system is simple, since the same files are accessed by all clients and simply replacing them with new files updates the data that clients access. [4] Until 1999, this was the primary method of distribution for most information over the Internet (excepting personal communications, newsletters, and other activities which lend themselves to e-mail). [4], [6]

Then, in 1999, a university student created Napster to help him find mpeg 1 level 3 (mp3, a music format) files Online. Napster is a centralized peer to peer network, meaning that users on the network share files with each other (taking on the roles of both server and client), but must connect through central servers to find out who is on the network to share files with. [6] The next theoretical advancement came in the form of decentralized peer to peer networks, where each user asks other users it knows if it has a certain file, and if those users do not, they in turn ask users they know, and so forth until the file is found or the request dies from not being answered (preventing messages searching for non existent files from crashing the network). Finally, by 2000, Ian Clarke had started Freenet, an adaptive, decentralized peer to peer network, where information moves itself around to different nodes depending on how much it is requested (so that information will have to travel less, increasing the speed of the network over time). [2]

All peer to peer networks have a flaw: there is no inherent way for information to be updated in them. Actually, centralized and decentralized (but not adaptive) peer to peer networks do make the client-server file replacement method possible, but are not usually employed since these types of networks are used for sharing commodity files, with many users having copies available for download of the same information. [6] Adaptive, decentralized peer to peer networks offer enough advantages in speed and in the facilitation of free speech to make them potentially better suited for general use than

client-server networks (like the World Wide Web), but lack any way to update information once it is inserted. [0] By implementing this paper's solution, adaptive, decentralized peer to peer networks can be made updatable, allowing them to be used for the general accessing of information.

Essential Theories and Practices

Adaptive, Decentralized Peer to Peer Networks

Foundation

An adaptive, decentralized peer to peer network consists of nodes that pass messages amongst each other, requesting and transferring data, moving information to nodes where it is commonly requested, yet with each node knowing about a few others to send messages to. [2] A sample topology is shown in Figure 0.

This topology is in contrast to that of traditional client-server networks, whose topology is centralized, as seen in Figure 1. In both figures, the spatial positioning of objects is intended to imply physical separations, but is not to scale.

Purpose

Ian Clarke, the inventor of adaptive, decentralized peer to peer networks, wanted to create an environment where information could be shared and completely free speech would be protected. To accomplish this goal, information would need to be completely anonymous, so that it would be impossible to tell who posted what, who requested what, and who has what. [1] In addition, Clarke realized that a fundamental problem with the Internet as it exists is that it pays no concern to geography, so that when certain information is requested, messages are sent through tens or even hundreds of computers over thousands of physical miles, with each exchange consuming precious milliseconds. The adaptive part of his network topology is to move information physically closer to where it is most often requested, reducing the latency of the transfer thereof. [0]

Nodes

Each node on the network knows only that its neighbors exist, as does its neighbors, and so on, so that all nodes are connected through each other without knowing about the existence of more than a few. This makes it impossible to tell which node has what information, since every message makes it look as though it originated from a neighbor, though that is usually not the case. In addition, nodes are black boxes, in that how one functions internally is of no concern to the network, so long as a standard set of messaging protocols is adhered to. [0] Finally, nodes do not have any knowledge of what information they have stored on them (or at least they should not), since that protects node owners from being liable for the information stored on their node (this usually means having the computer internally encrypt the information to keep it secret). [3]

Hashes

Hash tables are one way to store information in a node and the means referenced to in this paper because of their simplicity. Other, more efficient and secure methods exist, but are not referenced to here for the sake of space. Modifying the solution should not require significant work, only a different program in the node to implement the updating process.

A hash consists of three key-field pairs: name, information, and TTL. (see Time to Live, page 3).

Messages

There are four kinds of messages in an adaptive, decentralized peer to peer network. The first is an insert message, which puts new

information into the network. The next is a request message, which ask for some information by name. A request reply sends the information asked for by a request message. Finally, a request failure message is returned if the TTL (see Time to Live, page 3) of a request message runs out. These messages are the only way nodes can communicate with each other and are designed to make node implementation as independent as possible. [0]

Date Storage

The adaptive part of these networks comes from the way that data is stored. After data is inserted at a node, it will move around to the modes where it is requested most often. For example, if a particular hash is located on a node in Europe but a lot of request for it are coming from a node in Nippon, the hash will be copied to the node in Nippon to make access faster. Similarly, if hardly anyone is requesting it in Europe, eventually it will be deleted on that node.

Time To Live

All messages have a Time To Live (TTL). This states how many times a message can be resent to other nodes before returning an error message. Each time a node resends a message, the message's TTL is decremented by one, until it reaches zero and the message fails. TTLs keep the network from overflowing with messages pointing to null hashes and insure that messages will not loop forever between a set of nodes. [0]

Cryptography - Public Key

Cryptography has been around as long as certain people have had an interest in keeping information away from others for a certain amount of time. As early as 500 BC, the Hebrews were using various methods of substituting letters for one another, decipherable only with the proper 'key', to keep messages secret. [5] In this century, cryptography has made major advances thanks to the processing power of computers. This century has also seen the rise of public key encryption, where two keys are used in sending encrypted messages: one to encrypt, the other to decrypt. The private key used for decrypting is kept secret, but the public key for encrypting is shared freely, since the private key cannot be determined from the public key. In addition, the private key can encrypt text that is said to be signed, in that the originator of the text can be verified with the public key (assuming the private key has not been compromised). [7] Public key encryption offers the advantage of avoiding flimsy protection through obscurity and a more secure method of encrypting plain text.

The Difficulty with Updating Peer to Peer Networks

Unlike in the client-server model, peer to peer networks offer no way to control directly the information that is on them. For centralized and decentralized peer to peer networks, it is not even really desirable, since their purpose is for individual users to share their personal files with each other in a swap meet, where each user shares and shares alike. [6] On adaptive networks, information is had through keys stored on unknown computers. Rather than trying to get the information that others have, the user wants to find certain information. Unfortunately for the updater, the distributed nature makes updating difficult, since it must be done not just once, but dozens, hundreds, or even thousands of time over on different nodes. [0]

Another problem with making hashes updatable is that they may then have identifying marks on them (see section 3.4). Thus, updating comes at the cost of anonymity. One of the main goals of Ian Clarke is to make a network where all information is as free as possible, and for him that means that it must be anonymous so that there is no threat of retaliation, which might prevent some

information from ever being seen. [1] Therefore, any updating solution must allow for information to remain anonymous if so desired by the inserter, though he will not be able to update it.

Also, it takes time for updates to be distributed, so it may take minutes or even hours before information is changed. Most of the time, this is fine, since the majority of Web sites, for example, are only updated once a week or less. Those sites that update all of the time, like news and discussion sites, would not fare well with such a lag. News sites would post stories, only to have readers learn of their existence once the stories are already hours old, preventing truly timely information from being shared. Discussions would become disjointed due to the lag and progress slowly, as participants waited for their nearby copy of the information to be updated.

A Proposal for Updating Adaptive, Decentralized Peer to Peer Networks

Overview

The process of updating requires a lot of infrastructure and the addition of extra fields in the hash table of each item in the network. The following proposal is not, though, the simplest way to update an adaptive, decentralized peer to peer network; that would be to send out messages that replace the old ones by using the same key, thereby overwriting old data. Such a system, however, would be very easy to crack and not very trustworthy (anyone can replace any hash).

Hash

Existing fields

The fields that already exist in the hash, the name, the information, and the TTL, will continue to function unchanged. The main difference is that the hash will now contain additional fields.

Password

To make the network updatable, first a special field for allowing updating is needed. This field needs to contain a password to allow for write access to the information, thereby letting only authorized persons make changes. To keep the password from easily being cracked, it should never be stored as clear text, but rather encrypted using a strong cipher (such as twofish) with a key known only to the node. A public encryption key should be stored in the update field, as well, so that it can be used to verify the signature on the update packet (explained in Update message, page 4).

Time

Although not necessary, a field containing the date and time of updates is useful for acknowledging how recent the information is. Basically, each hash has a time field, consisting of a two-dimentional array of time stamps. The time stamps are paired by the time that an update message (or the original data insert message) was sent and when the hash was actually updated.

Update message

Just like any message on an adaptive, decentralized peer to peer network, an update message contains a TTL. Next, a name is stated, so that it can find the hashes on nodes that it wants to update. A password is also necessary, so that write access can be obtained to the hashes it is trying to update. This password is encrypted while it is in transmission so that it cannot be cracked, but once on the node it might not be necessary to remain as such. Also, the message should be signed with a private key that matches the public key stored in the hash to validate the update message. While the system could work without the validation, it makes it more secure by reducing the chance that an unauthorized entity who has obtained the password for a

hash could make use of it. Finally, the time at which the message was first sent is stored.

Most importantly, the update message contains information. This comes in one of two forms: either a replacement or a patch. The former completely overwrites the information in the hash, just as if a data insert message were sent, while the latter contains instructions on what parts of the information to change. The patching is accomplished using a standardized patching method (such as the one employed by the diff and patch programs found on most Unixish systems) that all the nodes have to make use of; deviations would lead to buggy patching at the worst and no updates at the best.

Updating

Get ready: it's time to review the life of a hash table in an adaptive, decentralized peer to peer network.

The hash's life begins when it is created with a name and information put in it. For this example, it will be called `MacHack` and contains information about the upcoming MacHack conference. It also has a password and contains the MacHack public key. It resides quietly on node A.

Then, as June approaches, many people around node B become interested in the conference and want to know more about it. Thus, they request `MacHack` so they can learn more. Node B doesn't know about node A, but it does know a node C, so it sends a message to node C, looking for `MacHack`. Node C doesn't have the hash, but node C knows about node A, so it asks node A for it. Well, node A has `MacHack` on it, so it sends a message back to C, which sends a message to B, containing a copy of `MacHack`. Eventually, `MacHack` is so popular that a copy of it is put on B to ease the load on the network and bring the information closer to those who want it.

The week before MacHack, it turns out that one of the speakers is sick and won't be able to come, so, it's time to update `MacHack`. First, the corrections are made and a patch is created. This patch is put in an update message and sent out to A (the node the MacHack folks are connected to) to all of the nodes it knows. Well, A knows about D and E, so it goes there. As it happens, E knows about node B, so it sends the message there after not finding a copy of `MacHack` on itself. D knows only about A, so the update message keeps bouncing between them until the TTL reaches 0.

Upon reaching B, a match is found. B checks the passwords against each other first to make sure that the message is authorized to make an update. Then, finding that the hash is signed, the signature in the message is verified with the public encryption key in the hash to further assure the message's authorization. Finally, the time on the update message and the last time stamp on the hash are compared, and if the update is newer than the last one applied (this prevents old messages still in the network from overwriting new ones), it updates the hash and then sends the message on. At each node, this process is repeated. Each time an update is applied, the message's TTL is reset. Each time is passes through a node without updating it will reduce its TTL by one. This way, update messages will die when no more nodes need updating or no more nodes contain the hash in question.

Back to `MacHack`, it is updated and people learn that the speaker in question will not be attending. They are sad, but are also glad that Gordon Worley has kindly agreed to fill in.

A couple days later, Gordon changes his mind, so another update is sent out. This one follows the same process as before.

A few days later, the first update to MacHack makes its way back to node B. Strange. At first, everything looks to be in order, but then B

notices that this update is older than the lastest one, thus it rejects it, subtracts 1 from it's TTL, and sends it on to eventually die out.

The day before MacHack, in an attempt at sabotage, the folks at BillHack, a competing conference, send out a false update claiming that MacHack was cancled because all of it's employees found that Windows was easier to use than the Mac OS. It has the same name and a more recent time plus, through some social engineering, even has the correct password. Yet, they couldn't sign it with the MacHack private key, but they send it out anyone, hoping for the best. When node B gets this update message, it starts to go through it and everything looks okay, but then it finds that the information's signature does not verify with the proper key. Uh-oh. This is a bad one. The update is rejected, it's TTL decremented by 1 and sent on it's way. It is not deleted out right, though, just in case node B made a mistake. Better luck next time, l4m3rz. ;-)

On a final note, in the event of a random error, the message just sent on it's way with it's TTL minus 1.

See Figure 2 for a flowchart of the updating process.

Security Concerns

In designing the updating process, the greatest concern for security has been taken. By using a password, the network employs a basic system that is generally sufficient to prevent cracking. Encrypting the password in transit between nodes, the chance of someone intercepting the password and using it to send falsified update messages is reduced greatly. With a private encryption key's signature in the update message and its verification with the public encryption key in the corresponding hash, most of the easy methods of sending fake messages are eliminated. All these measures do not, however, prevent fundamental security issues which may exist within the design of the network or the nodes.

One issue is the spread of virii and worms. If a malicious update message were coded to patch anonymous hashes to contain viral code, the results could be devastating for users. Someone expecting to download a copy of a research paper or an important historical document could instead end up with lost files or system failures. The best ways to combat this problem would be to scan all anonymous hashes for virii before reading their information and to try to download only hashes that have been verified as authentic. Virii should not be any more common than they are through any other Internet protocol, but the constant movement of information to lots of different nodes increases the chances of infection.

Of greater concern, though, is the potential loss of the anonymity that adaptive decentralized peer to peer networks provide. In particular, the verification process requires a public encryption key which is just that: public. By it's nature, the public key is meant for use by any sender to secure the contents of some text to the recipient with the private key. This is the reason why this feature is optional and left up to the node. It would be preferable if the nodes verified the signature if present, but if not allow the update to proceed anyway in the interest of keeping hashes as up to date as possible. It just depends on whether a particular node wants to contain only verified hashes or anonymous ones as well (at least in the sense that they can be updated).

Applications

In general, adaptive, decentralized peer to peer networks are a boon for free speech. All information can be completely anonymous, so that no one can track an individual user or group of users. Under such conditions, previously unknown opinions and facts forced into censorship by fear of retaliation from the government or corporations would be made

available to the world. In such an environment, it would be harder for people and organizations with the legal power of force to exploit or harm the populous which they are supposed to serve. Such a system of checks and balances would lead to a freer society that does not infringe upon the natural rights of a person.

By making this information updatable, it will be more useful to a general audience (i.e. it can be used for more than file swapping). For example, if inaccurate information was previously inserted into the network, replacing it with an update should make the appropriate errata, improving the value of the information and the network as a whole. With good information, the network would better the world, by making accurate facts easily available to anyone on many topics, even those that are considered taboo or censored by different governments around the world for various reasons.

Updatability does have the downside of possibly introducing a means of censorship if the updating system is not secure enough, so that update messages could be sent out to eliminate existing information. Also, the creator could censor his own work, or be forced to do so against his will. For instance, a hacker might post the API for a proprietary piece of hardware that he reverse engineered (which is legal under United States law). The company that creates the hardware could then come to him and force him, under threat of law suit (in spite of the law, the hacker could not afford the basic court costs), to overwrite his information. He could always repost anonymously (or could have done so to begin with), but then the incentive to do the hack is removed, since he will not be getting any credit for it. Similar actions could happen to artists with controversial works, the politically oppressed, or any other person trying to provide information that someone else would prefer be kept in a tiny, locked box. Hopefully, the network would not be used to spread propaganda and propagate false facts, since the majority of its users would be interested in having a useful network, so they would put factual information into it.

Conclusion

By implementing the solution described, an adaptive, decentralized peer to peer network can be updated. To do so, techniques like public key cryptography and patching must be employed to make the system secure and efficient. Once updatability is achieved, free speech and better, more accurate information will be readily available to the Internet using public. Such a system has never existed before, though, so it is unknown how users will treat their new found power to express themselves freely, without fear of retribution. In the past, systems like Usenet and Web logs with message boards, like Slashdot, have been plagued with spam and trolls, but those systems are about discussing with other people, not sharing information (though the former often facilitates the latter). Until this solution is implemented in a real world network, such as Freenet, the answers to such questions will remain purely hypothetical.

Bibliography

[0] Clarke, Ian. "A Distributed Decentralized Information Storage and Retrieval System". Freenet Project, The. 1999. <http://freenet.sourceforge.net/freenet.pdf>. (15 April, 2000).

[1] Clarke, Ian. "Freenet - Censorship and Copyright". Freenet Project, The. <http://freenet.sourceforge.net/index.phppage=philosophy>. (22 Aug. 2000).

[2] "Freenet - Frequently Asked Questions". Freenet Project, The. <http://freenet.sourceforge.net/index.phppage=faq>. (22 Aug. 2000).

[3] Langley, Adam, Scott Miller. "Freenet - Crypto Layer". Freenet Project, The. <http://freenet.sourceforge.net/index.phppage=fncrypto>. (22 Aug. 2000).

[4] Segaller, Stephen. Nerds 2.0.1: A Brief History of the Internet. New York: TV Books, L.L.C., 1998.

[5] "SSH - Cryptography A-Z - Introduction to Cryptography". Cryptography A-2-Z. 1999. <http://www.ssh.fi/tech/crypto/intro.html>. (1 Aug. 2000).

[6] Yamamoto, Mike and John Borland. "A brave new--or old--world?". The P2P Myth. 26 Oct. 2000. <http://news.cnet.com/news/0-1005-201-3248711-1.html>. (28 Oct. 2000).

[7] Zimmerman, Phil, et al.. Intro to Crypto. United States: Network Associates, Inc., 1999.

Works Consulted

Elgarten, Gerald H., Alfred S. Posamentier, Stephen E. Moresh. Using Computers in Mathematics. Menlo Park, California: Addison-Wesley Publishing Company, 1983.

Lutz, Mark. Programming Python. Cambridge: O'Reilly & Associates, Inc., 1996.

Oualline, Steve. Practicle C++ Programming. Cambridge: O'Reilly & Associates, Inc., 1995.

Stephenson, Neal. Cryptonomicon. New York: HarperCollins Publishers, 2000.

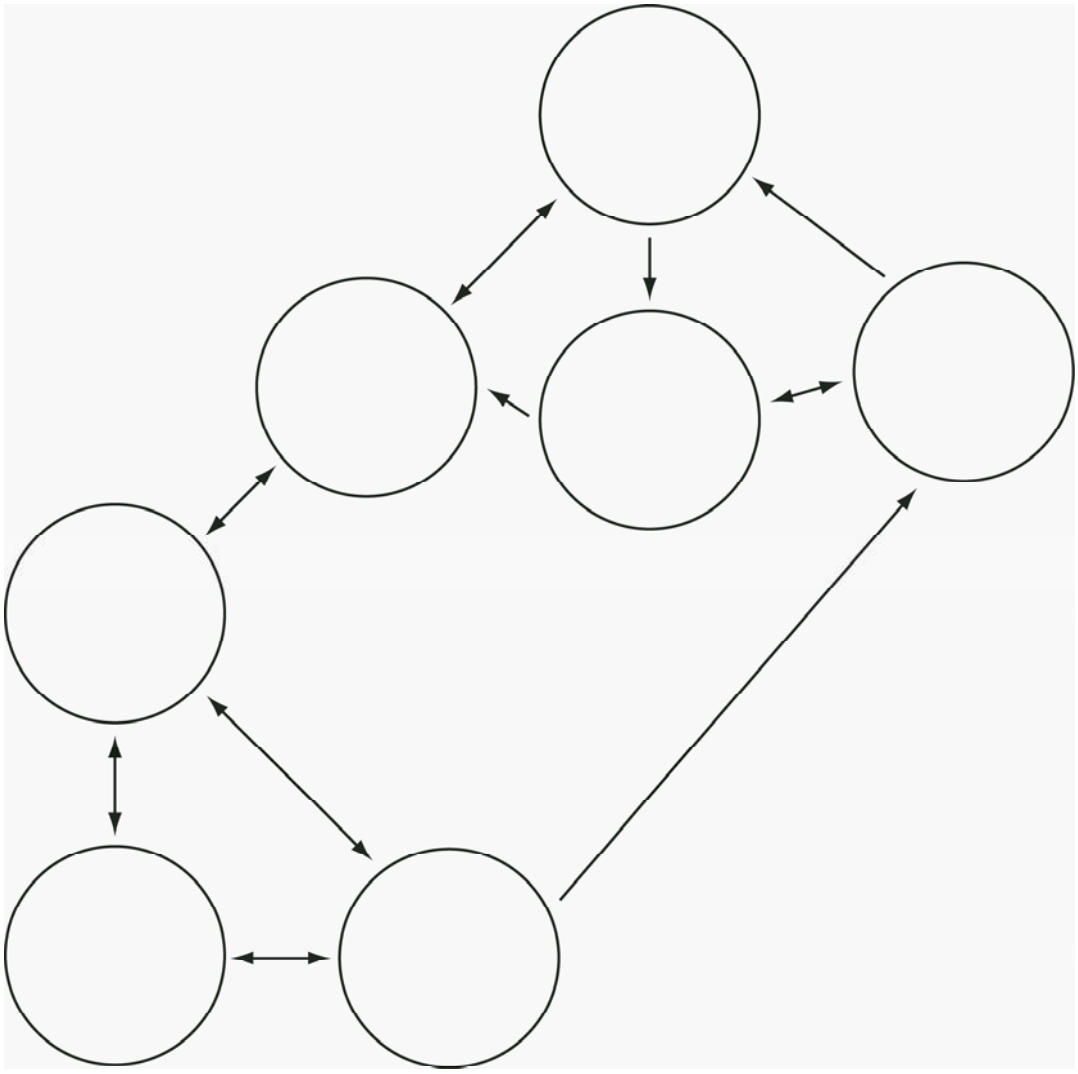

Figure 0. Sample adaptive, decentralized peer to peer network topology. Circles are nodes and arrows represent the direction of knowledge (i.e. in which messages are sent).

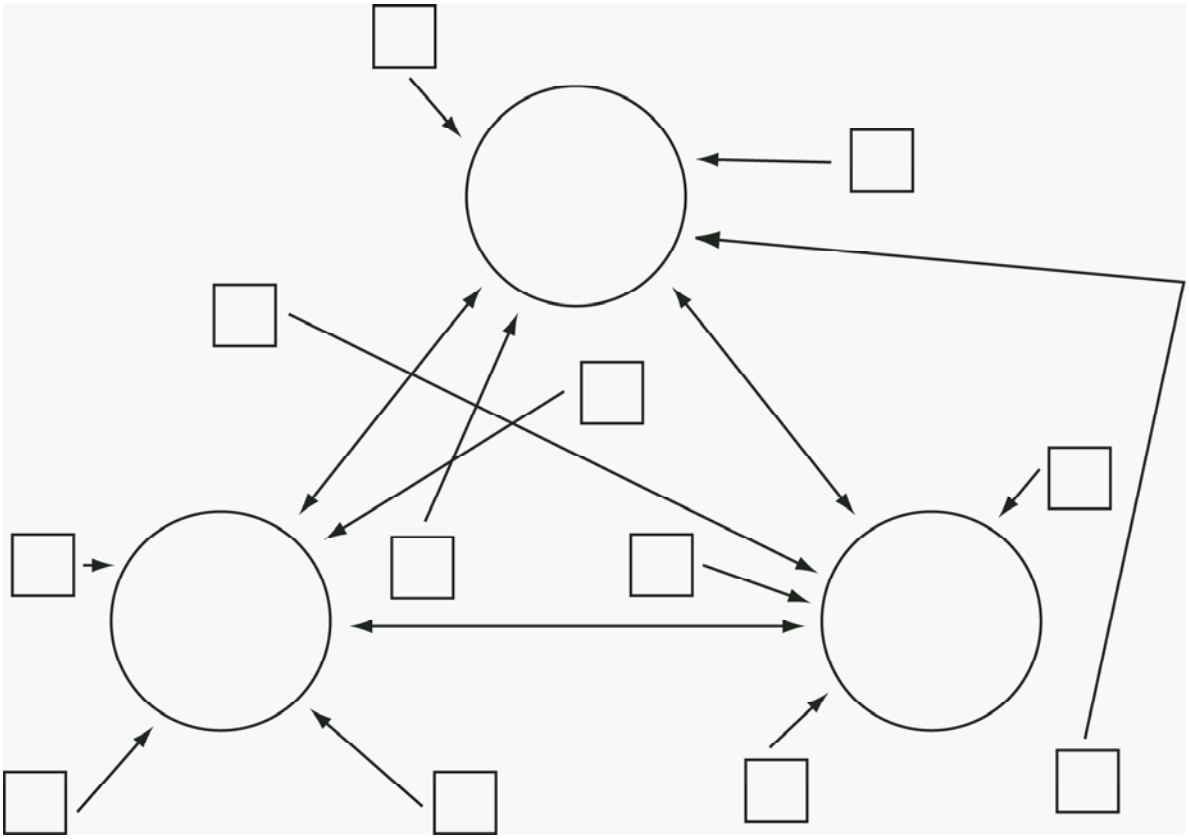

Figure 1. Sample client-server network topology. Circles are servers, squares are nodes, and arrows indicate direction of information requests.

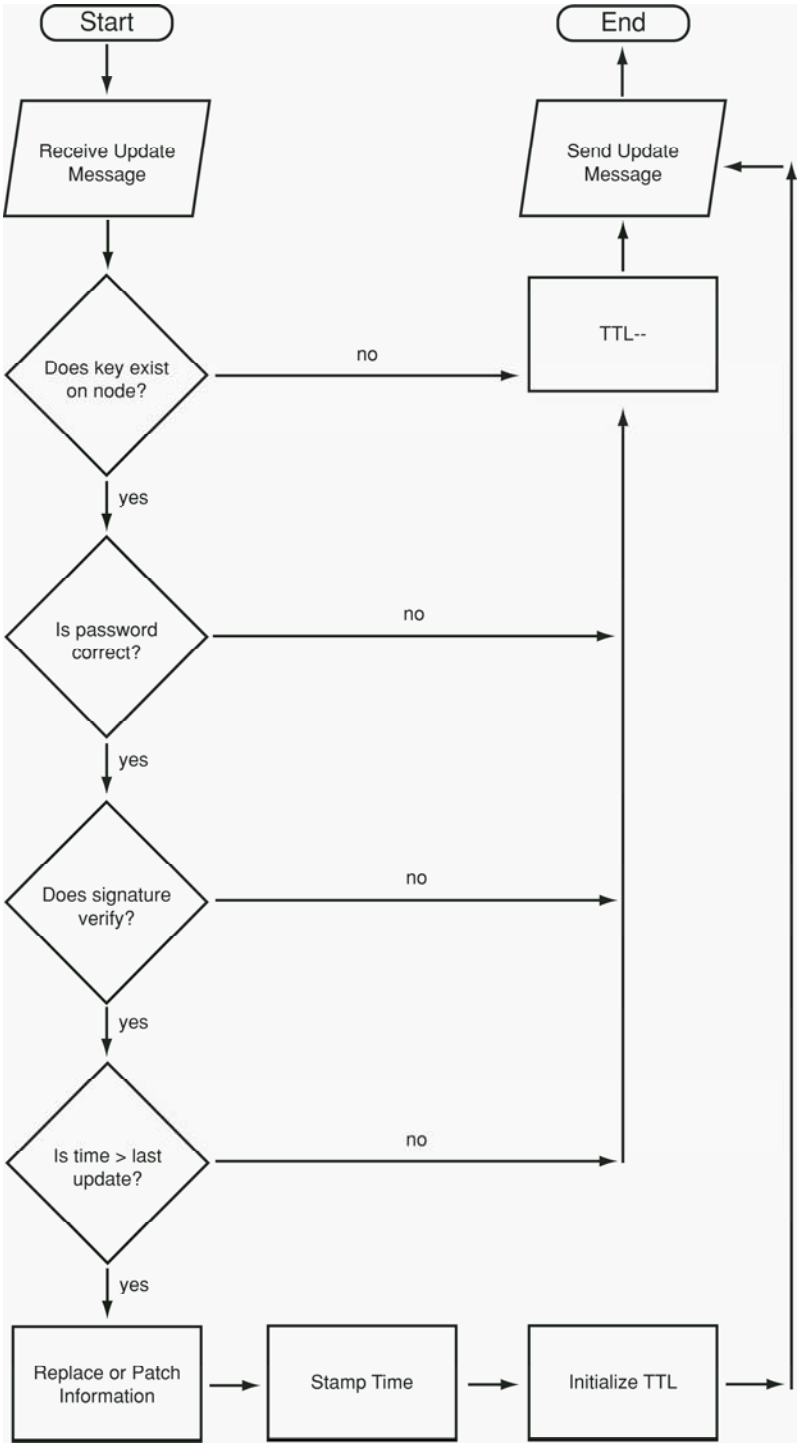

Figure 2. Flow chart of the update process within one node In the event of any errors, the process jumps to the TTL-- block.