Bypassing Firewalls Using A Negotiation Server and Port Prediction

Simplifying Peer to Peer Networking Applications for the End User

Benjamin Stroud
CS526
Advanced Internet and Web Systems
Spring 2010
bstroud@uccs.edu

Abstract

Peer to Peer networks have been shown to be a powerful cost savings tool for any content distributor. By minimizing centralized connections to the distributor and offloading much of the bandwidth requirements onto the end users interested in the content distributor are able to minimize instantaneous bandwidth spikes as new content becomes available and many users want a copy all at once. The peer to peer solution alleviates the peak usage charge that most content providers pay to their ISPs every month. One major hurdle to the success of this cost savings approach is that most end users are now behind NATed firewalls and must take deliberate action such as opening a range of ports in order to allow unsolicited connections into their internal network. If this step is not taken the user is now simply a consumer of content who is contributing nothing back to the network as a whole and the burden on the originator of the content is increased. In this paper I present one solution to this problem that was developed during my time at Network Foundation Technologies (NFT). I will first give some basic background on the problem, and discuss some previous solutions. I will also give an overall view of NFT and what particular problems we were trying to solve when we developed our solution. In the subsequent sections I will discuss the details of our particular solution, its success rates, and what issues still have yet to be overcome. I will also discuss other areas of computing where this technology could be beneficial. Finally I will present my overall conclusions about the solution and discuss future research that can be done.

I. INTRODUCTION

Peer to Peer network based software is often seen as a daunting concept both for the software creator and for the average user. Our goal in computing has often been to take seemingly complex tasks and develop tools to simplify the execution of those tasks to the point where their every day use is taken for granted. In many ways this is the goal of the solution presented in this paper. When successful this technology simplifies the participation in peer to peer networks for the end user and brings the software that uses it to a point that it "just works" for the user without a lot of difficulty on their part. In the following sections I will present some basic background on the problem and its current state.

II. THE PROBLEM

NATted Firewalls generally don't allow unsolicited connections from the outside internet to enter into the “protected zone” of internal machines that are connected to the firewall [2]. This is a major roadblock to peer to peer applications because the peer to peer model is built around the idea new nodes (or clients of the network) will be able to join to the most suitable peer who is already established within the network. For this to happen the established node must allow the new node to connect to it at any time, as a client would connect to a server in a traditional network model. If the established node is protected by a NATed Firewall then the unsolicited connection being attempted by the new node will be blocked, unless the user of the established client takes some deliberate action such as opening ports that are known to be available to the entire network.

This deliberate action is often beyond the willingness or capabilities of the average computer user. Many people don't even realize, for example, that the router they use to gain wireless access around the house is actually also a NAT device as well as a firewall. If the established node is unable or unwilling to allow connections in, then that node is simply a consumer and is contributing nothing to the network which increases the burden on the established nodes and the server from which the content originates. If most of the nodes participating in the peer to peer network do not contribute back to the network then the bandwidth offloading benefits of the network are lost and the peer to peer network functions more like a traditional client server based network.

In addition to the problem of unsolicited connections being blocked, certain NAT devices will also offset the port that a client will request data over, and what is transmitted to the receiver. This was done in part as a countermeasure to some of the previous solutions explored in the following section. This further complicates the problem of establishing peer to peer connections through consumer level routers. An approach to overcoming both of these issues is discussed in section IV.

II. PREVIOUS SOLUTIONS

A) UPNP

UPNP or Universal Plug and Play in the realm of hardware firewall devices was a collaborative effort between software developers and hardware manufacturers to try and build a mechanism by which firewalls could be programmatically controlled via an application running on a system within the protected network [4]. The idea was that an application would be able to open ports without any action taken by the end user. The key requirement and its ultimate flaw was that hardware manufacturers had to make their products compatible with and configurable by this protocol. We found through experimentation that the majority of device manufacturers were either unwilling to support this feature or that it was off by default and users had to activate it manually. We assumed that this was because of security concerns over potential misuse of the protocol by malicious persons or organizations. I discuss our results with UPNP in a later section.

B) UDP Hole Punching

A UDP based solution for NAT traversal is often called “Hole Punching”. The basic algorithm is similar to the solution presented in this paper but did not account for the port offsets (described above) that were later introduced. The algorithm generally works as follows: A and B are two nodes, both behind NAT devices. A wishes to make a connection to B but cannot because A does not know B's internal address or port mapping. A and B instead make connections to the server known to both clients and reports their internal addresses and port mappings. Once A and B both know each others complete addresses and port mappings they can connect directly to each other. The classic version of this solution is able to traverse the NAT, but does not account for the blocking of unsolicited connections or port offsets that were introduced after this technique became popular [1].

STUNT

STUNT or Simple Traversal of UDP Through NATs and TCP too, is in theory very similar to the solution presented in this paper, but is designed with TCP in mind. STUNT extended basic principle of NAT traversal established by UDP hole punching is extended to TCP. It also attempted to overcome the blocking of unsolicited connections as well as the port offset problems discussed above[2]. However at the time that NFT was attempting to resolve these problems itself, the STUNT implementation was still immature and mostly theoretical. We had multiple problems using the researchers test applications and the results that we found with the test devices we used were not as encouraging as we had hoped. We decided to build on the basic principles used in STUNT but create our own specific implementation.

III. A SPECIFIC SCENARIO

In this section I discuss one particular software development company which experienced the problem of dealing with the increasing number of NATed and firewalled nodes that wished to participate in a distributed, peer to peer network but were unable or unwilling to manually configure their environment to allow other users to make unsolicited connections to them. The company was Network Foundation Technologies (NFT) [3].

NFT is an online television broadcaster that specializes in live video broadcasts over the Internet. The company's primary intellectual properties are patents that deal with distributing real time content over a peer to peer style network of nodes who are consuming the content at the same time that they are redistributing it to other nodes in the network. This particular network model uses a balanced binary tree as the conceptual model of the network. What this means in practical terms is that two client nodes will connect directly to the origin server that is responsible for packetizing and broadcasting the real time content. These two nodes theoretically represent the only bandwidth cost to the broadcaster. These two nodes, once connected to the server and are receiving the data stream then allow two more nodes to connect to each of them, creating the second level of the tree which contains four nodes and so on, each level increasing by 2^n nodes where n is the level of the tree. Each new node joining is added as a leaf to the existing tree [5]. A visual representation of this networking model is contained in Figure I.

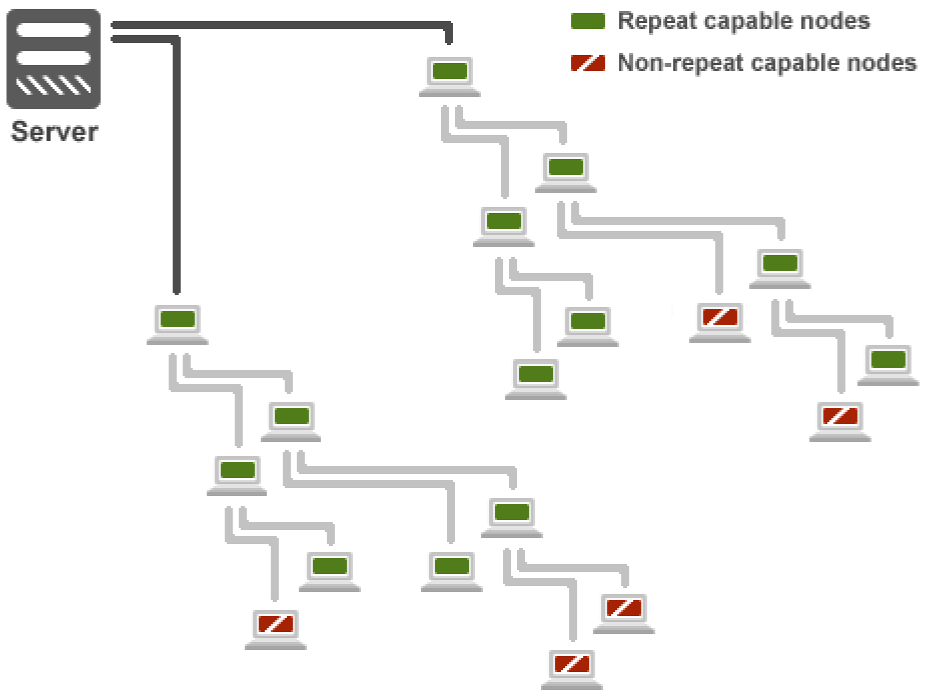

Figure I: NFT Distributed Network [3]

There are two key assumptions about nodes entering the network. The first is that the node has enough upstream bandwidth to redistribute the stream to two other nodes, essentially the node is responsible for uploading twice what it downloads. The second assumption is that the node will allow unsolicited connection to it from other unknown nodes as they join the network. Any node who is not able to meet these two requirements is remains on the edge of the network and is designated as a non-repeater so that no incoming nodes will attempt to connect to it [5]. The challenges of the first assumption are beyond the scope of this paper. The second assumption is where the company encountered the problem described in Section I.

For the first year or two after the initial product release the Chief Scientist and the lead developers operated under the assumption that either technologies such as UPNP [4] or user action would open firewalls and forward ports appropriately. This, as many peer to peer based companies are discovering, was not the case. In reality only about ten percent of nodes were able to allow unsolicited connections to be made to them. Using the binary tree model meant that only 20% of the network was supported in a distributed fashion. In general terms each new level of the tree accounts for about 50% of the total network so an addition of 10 repeaters to the network means that 20 non repeaters can be supported, for reasonably sized networks.

After several months of attempting to create instructional websites directing uses to open ports manually, we decided to begin researching reliable methods for bypassing firewalls that would work for our specific circumstances. After reviewing much of the previous work discussed in the preceding section we were able to devise a somewhat novel solution to the problem which is presented in the following section.

IV. A SOLUTION

Our solution combined the idea of UDP based hold punching which overcomes the problem of blocked unsolicited connections with a negotiation server that facilitated port prediction which helps overcomes the problem of offset ports. An overview of the solution is presented in Figure 2. I discuss the details of the solution in the following sections.

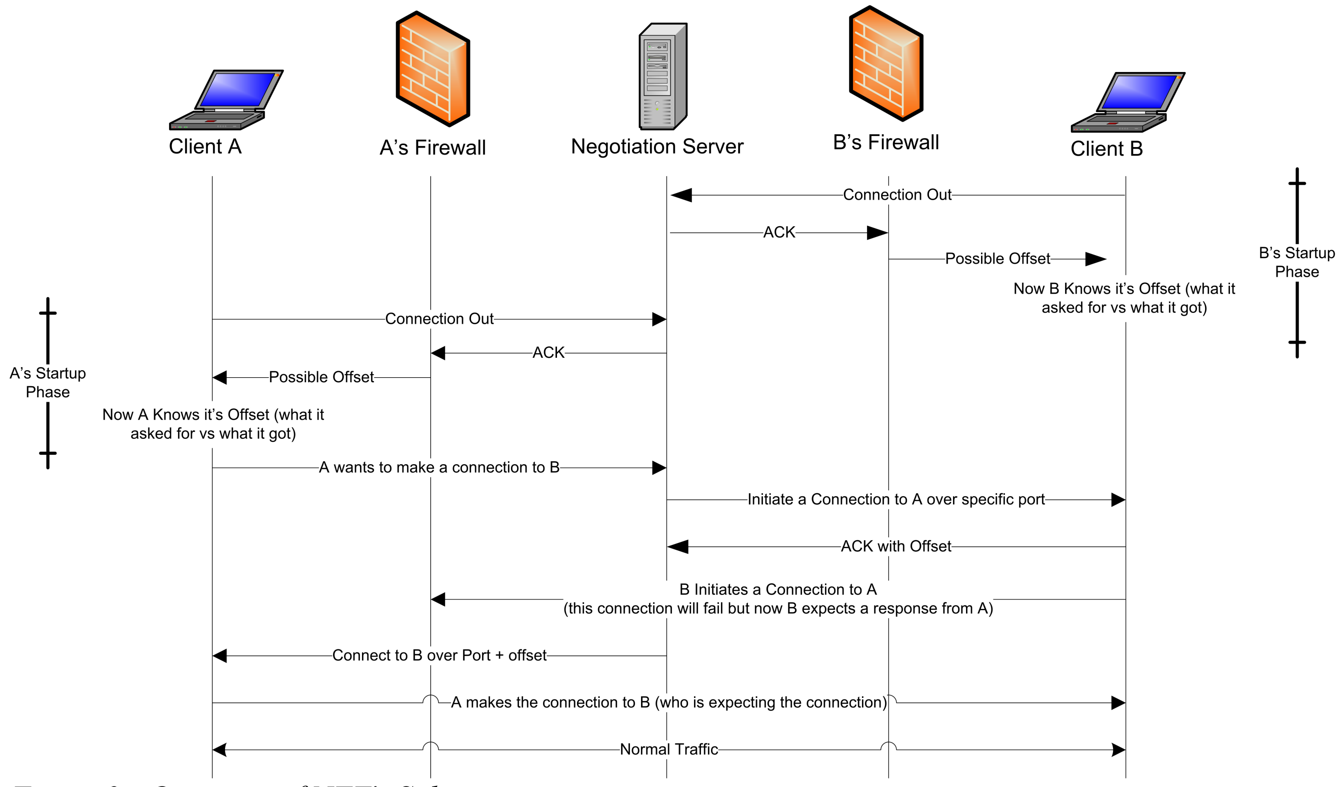
Client A
A's Firewall
Negotiation Server
B's Firewall
Client B
Connection Out
ACK
Possible Offset
B's Startup Phase
Now B Knows it's Offset (what it asked for vs what it got)
Connection Out
ACK
A's Startup Phase
Possible Offset
Now A Knows it's Offset (what it asked for vs what it got)
A wants to make a connection to B
Initiate a Connection to A over specific port
ACK with Offset
B Initiates a Connection to A (this connection will fail but now B expects a response from A)
Connect to B over Port + offset
A makes the connection to B (who is expecting the connection)
Normal Traffic

Figure 2: Overview of NFT's Solution

A) How to Interpret the Figure

In this figure time progresses from top to bottom. Each vertical line represents a potential receiver or transmitter of network traffic. Each horizontal line represents traffic being sent between two endpoints, with arrows indicating who is receiving and who is receiving and who is transmitting the data. Some simplifications in the process have been made to aid in understanding. The overall steps are preserved.

B) The Scenario

In this scenario both Client A and Client B home users who are behind a NATed Firewall device. The devices that A and B are behind allow any traffic that is outbound, but filter inbound traffic. The Negotiation Server is on an open network with no filtering devices in place. Client B is first to join the network and has already established a connection to it when A decides to join. Client A will attempt to connect to Client B and receive a data stream over the connection. Client B has no prior knowledge of Client A before the connection is attempted. The address and listening port of the Negotiation Server are known to all the members of the network as soon as they are fully initialized and running and before any connection attempts are made.

C) The Start-up Process

Before any connections are attempted to nodes within the network all nodes must first make contact with the Negotiation Server. This basically happens as follows. The node makes a connection out through its firewall device to the Negotiation Server which contains its known internal address, which is allowed through since we assume there are no restrictions on outbound connections. The Negotiation Server sends an acknowledgment(or ACK) back which includes the clients external address. Now that the client knows both its internal and external addresses it can establish any port offsets that might be used by its filtering device. Based on the difference between the port that the client made the request on and what it received back, an offset is established and saved by the client. This back and forth exchange can be done multiple times to establish any offsetting patterns, such as no offset, simply a static offset, a predictably changing variable offset, or a random offset. One of the weaknesses of this solution is in dealing with random offsets. I will discuss this more in later sections.

D) The Peer Connection Process

At this point both Client A and B have completed their exchange with the Negotiation Server and established their respective offsets and full addresses. In addition recall that Client B has already joined the network, is receiving the stream, and is available for connection attempts from other nodes. Client A now attempts to join the network using Client B as a peer.

First Client A informs the Negotiation Server that it would like to make a connection to Client B. Client A is now in wait mode for a response from the Negotiation Server. Meanwhile the Negotiation Server instructs Client B to connect to to Client A. Client B responds with an ACK that includes its previously discovered offset and internal address. Client B then attempts a connection to Client A as instructed, which will be blocked by A's firewall since this connection from B is unsolicited (A has yet to attempt a direct connection to B). What this connection attempt on B's part does is it causes B's firewall device to expect a connection back from A that will now be a solicited connection that should be let through. Once A receives the signal from the Negotiation Server that it can now attempt a connection to B using the given offset, A will make the connection to B. This connection should be allowed through B's firewall since it is a solicited connection. Once this connection is made the two nodes can transmit data normally without the intervention of the Negotiation Server.

E) Implementation Considerations

1) Robustness of the Negotiation Server: This solution requires the Negotiation server to be running at all times for the network to continue to grow and change as nodes enter and leave. If at any point the Negotiation Server becomes unavailable, either due to an outright failure, or by being overwhelmed by connections, new connections will no longer be made. Because of this fact, the Negotiation server must be able to handle a heavy connection load and continually run for very long periods of time. For our Negotiation Server we used overlapped I/O as the mechanism for allocating and managing sockets [6]. While this was a Windows specific implementation and restricted us to a Windows environment, we were able to simulate about 60,000 clients making requests every 3 seconds to a single Negotiation Server for months at a time. The hardware that we ran our Negotiation Server on were fairly low end blade servers(dual core 2.4 GHz) running Windows XP.

One technique that helped improve the performance of the Negotiation Server was to pre-allocate a very large pool of unbound sockets at the beginning of its execution. The sockets would then be returned to the pool after they were released. This insured that the Negotiation Server could react to a large sudden surge in activity.

2) Protocol Considerations: Our original implementation before port prediction was added used only TCP as the network's protocol. Because of this, the higher level networking configuration algorithms relied on the guaranteed delivery of TCP data. The previous solutions discussed above used the UDP protocol to implement the hole punching portion of the algorithm which allows nodes to make essentially unsolicited connections to each other through Firewalls. Rather than attempt to re-design the hole punching algorithms around TCP we decided to use a hybrid protocol called UDT. At its lowest level it is really UDP with some of the guaranteed delivery features of TCP [7].

These considerations were specific to our needs at the time and are not necessarily required for a successful implementation of this concept. I leave it up to the designer to draw his or her own conclusions about which implementation choices make sense for their application.

V. SUCCESS RATES

After the initial implementation and testing phases were complete, we found that we were able to increase our population of "connectible" nodes from about 10-15% under our original manual solution to about 95% under this new more automated, negotiation based approach. Clearly our solution was a success.

Within this 5% of uncorrectable nodes there were two types. One type would consistently fail the port prediction portion of the connection algorithm because the offsets chosen by the filtering device were random and therefore unpredictable. The second type would sometimes succeed in making unsolicited connections, but would then break the connection or fail subsequent connection attempts. We theorized that these users had such lossy connections that connections under any scheme would be just as unpredictable and fragile. We were never quite able to pin point what portion of the 5% was due to which issue. Our solution still suffers from both problems, unpredictable, random offsets as well as difficulty functioning under extremely high latency networks. These would be interesting areas for future research as discussed in the concluding section.

VI. APPLICABLE AREAS OF COMPUTING

The particular implementation described here was used to facilitate a Television like broadcast over the open internet, sometimes referred to as IPTV. The primary driver of this implementation was not the activity of distributing the content, traditional server based implementations have been successfully doing this for many years, but rather the desire to save the bandwidth costs of the broadcaster by offloading traffic onto a distributed network that consisted of the users currently watching the feed. This idea of content distribution by the interested parties rather than a central server is not new, but the manner in which it is being done and the fact that it does not require explicit user intervention is. This new way of forming connections will be applicable to many areas of computing that strive to, or could benefit by building a decentralized peer to peer based network.

One such area where similar solutions are being tried is VoIP. It is advantageous for the service provider to setup direct connections between users wishing to communicate with one another rather than funnel all the traffic through a central server. Any centralized system will be quickly overwhelmed by a popular service such as Skype.

BitTorrent clients are another group that could benefit from this technology. Currently most applications require users to manually open ports on their devices in order to contribute to the network. This type of solution could greatly increase the speed and ease which with content is distributed.

Server-less Chat clients are similar to VoIP solutions, but are purely text based. Anyone interested in private or secure communication will be interested in eliminating the need to connect to a centralized chat server in order to have a conversation with a particular person.

Distributed Computing Systems could benefit from this technology with the idea that more casual home users could participate in solving a problem collectively before sending results to the central server. This could greatly reduce the cost and the load born by the central server and increase the efficiency of the network overall.

Online Multiplayer Games where users must exchange data would not have to rely on a central server except for an initial connection setup as described above. This could reduce the overall cost of maintaining such a game for the developers.

These are just a small sample of the areas where this research could decrease cost and load on centralized, or quasi distributed systems. Many networked, multi-user systems will gradually move in this direction and the impact of this kind of technology will continue to increase over the coming years.

VII. CONCLUSIONS

In this paper I have presented a viable and proven solution to the problem of the blocking of unsolicited connections from one peer to another if conscious user intervention is lacking for whatever reason. I have explored some of the previously proposed solutions to this problem as well as presented a current real world solution in detail. We have seen that this solution has a high rate of success in the volatile and unpredictable realm of the open Internet. I have discussed some of the implementation details to be considered when implementing this solution. We have also seen some other applications where this technology would be very beneficial. In the following section I discuss some unanswered questions as future research that could be done regarding this solution.

While this solution is a demonstrably viable one to the problems discussed in this paper, its success rate will likely begin to decrease as device manufacturers learn of these types of practices and attempt to block this behavior. It is my belief that this will come in the form of more and more devices using random offsets which will defeat the algorithm in its current incarnation. A similar trend emerged to combat the older form of UDP based hole punching. This trend was the inclusion of port offsets, which this solution accounts for. Until an agreement is reached between peer to peer content distributors and security minded firewall device manufactures each party will continue to adapt to the others activities in order to reach their own individual goals. I think the consumer as a whole would be better served by a cooperative solution.

VIII. FUTURE RESEARCH

As mentioned in a previous section, we were never able to satisfactorily determine what portion of our failed connections were due to random offsets and what portion were due to high latency connections. It would be interesting to attempt to quantify this proportion by using some sort of independent measure of the speed of packet transmission, both from client to client, to ascertain if the hops between were bad, or from individual clients to a local known good server to ascertain if the individual node itself is at fault. The percentage of nodes giving random offsets could be known simply by collecting the offset reports of the individual clients and determining which were random, or at least random enough to defeat the algorithm. Once these two quantities are known it may expose other categories of failure types that are yet unknown.

Another area of useful research would be not just attempting to quantify but actually deal with high latency or random offset nodes. Solving these two problems would increase the success rate of the algorithm and strengthen the network overall. The issue of high latency networks will likely decrease in significance as time goes on and networks become faster and more robust. However the ability to deal with random offsets will become more and more important as device manufacturers become more aware of this technique and introduce random offsets to specifically combat it.

IX REFERENCES

[1]Ford B., Srisuresh P., Kegel D. (2005)“Peer-to-Peer Communication Across Network Address Translators” brynosaurus.com Last Accessed 5 May 2010. http://www.brynosaurus.com/pub/net/p2pnat/

[2] Guha, S., Francis P. (2005) Characterization and Measurement of TCP Traversal through NATs and Firewalls. *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement : 18-18*

[3] “NFT Company - Technology” NFT-TV.com. Last Accessed 25 April. 2010. http://www.nft-tv.com/about.php?section=technology

[4] North, K. (2003) “Programmatically Controlling a UPnP-Capable Firewall” Last Accessed 4 May 2010. http://www.knoxscape.com/Upnp/NAT.htm

[5] O'Neal, M., Kleinpeter, J. (2005). Systems for distributing content data over a computer network and method of arranging nodes for distribution of data over a computer network. Patent 7543074 Issued on June 2, 2009.

[6] “Synchronization and Overlapped Input and Output” MSDN. Last Accessed 2 May 2010. http://msdn.microsoft.com/en-us/library/ms686358(VS.85).aspx

[7] “UDT: Breaking the Data Transfer Bottleneck” UDT Project Site. Last Accessed 30 April 2010. http://udt.sourceforge.net/