**The Use of Peer to Peer Networks for Production Level Applications**

Logan Esch

*Department of Computer Science*

*Montana State University*

*Bozeman, MT 59717, USA*

logan.esch@msu.montana.edu

**Abstract**

Peer to Peer networks have become infamous in recent years for their facilitation of copyright infringement. Their design allows for the rapid dissemination of files to numerous parties throughout the connected network, with a self-scaling architecture that reduces IT overhead. With their rising preeminence, peer to peer networks are being used for everything from hosting online video games to relaying video communication. This paper reviews their use by different organizations in their production environment and provides a comprehensive view of the best practices to their design, use and rational that makes a peer to peer approach appealing. It is written for a technically proficient audience that may be interested in how their own businesses or personal projects may adapt and take advantage of a peer to peer topology.

**1. Introduction**

To download files in a traditional client-server architecture, all clients connect to servers that are maintained by the organization that owns the files, and then proceed to request the files that they need. This design has properties that place a lot of strain on the servers themselves and the infrastructure supporting them, because they have to serve every client individually. Given the reasonable assumption that the core of the internet is much faster than the fringes, the minimum speed at which a client-server architecture can serve all clients desiring a file is at least equal to the maximum of either the file size times the number of clients who want that file divided by the speed at which the server can serve the file or to the file size divided by the minimum download speed of each individual client. That is the download speed for client-server architecture:

[1]

Where *N* is the number of clients who want the file, *F* is the number of bits in the file, *Us* is the upload speed of the server and *Dmin* is the minimum of all the clients download speeds. Looking at the given equation, it is easy to see how when the number of clients grows the bottleneck becomes how quickly the servers can upload the file, as the amount of work will grow linearly. This means that each client’s performance will suffer in direct proportion to how many clients there are.

Now in production for most applications it is quite rare for just one server to be handling all the traffic, oftentimes there are clusters of hundreds or even thousands to serve all the clients. But demand for any given file can be very sporadic, leading to wasted resources of having hundreds of idle servers, or overloaded servers when demand suddenly spikes. While there are mechanisms for powering down servers when idle and bringing them back online when needed, this cycle is difficult to setup, manage, troubleshoot and servers still take minutes to spin up and start distributing content. This delay at worst can mean that the few servers who were left get overloaded and crash and at best means that all clients will have sub optimal performance, leading to frustrated customers. Improving the hardware in use in the servers can help but once again these gains will be diminishing compared to their cost; jumping up a processor bin can double the cost and only give a 10% performance boost, doubling the ram can quadruple the price, and improving the hardware capabilities on any level will only serve to increase the waste of idle instances. Even if the production clusters were state of the art behemoths, their performance would still be determined by their internet connections link speed, which can be very expensive. Bills from ISPs tend to grow exponentially. What this all boils down to is this: in order to maintain satisfactory performance in a client-server architecture, organizations are going to have to spend lots of money on hardware, internet connections, and designing and troubleshooting some complex system for managing idle instances.

Peer to Peer (P2P) networks lack these problems. In a peer to peer network, clients can download the files they want from each other: each client is also a server. There is still one initial server that seeds the file, but instead of serving the entire file to each client as in the client-server architecture, the server will only have to upload the file to the network once, giving different portions to each client. The clients will then share their portions of the file with each other until they all have the complete file. This means that the download time on a P2P network is:

[1]

Where once again *F* is the file size, *us*is the server upload speed, *dmin*is the minimum of the download speed of the clients, but also where *ui* ­is the upload speed of each client. Notice that compared to the first equation the *NF* has been shifted to be split over the combined upload speeds of the clients. This means that as more clients connect and start downloading a file, more clients also start serving the file; peer to peer networks are self-scaling. Instead of the total download time growing linearly as demand increases like a client-server architecture, P2P networks are logarithmic. P2P networks will also have a fixed cost as demand grows because the load on the server owned by the organization will remain constant irrespective of the number of clients. It doesn’t matter if ten clients want a file or ten million, the server still will have the same amount of work. These advantages are why many organizations like Skype, BitTorrent and ‘tisu use peer to peer solutions: they are faster and cheaper than their traditional counterparts.

The advantages of P2P systems can be leveraged to address many problems that client-server solutions currently solve at great expense. Peer to peer solutions have been successfully implemented to address real time communications, file sharing, and online gaming [1, 3, 6]. The most successful of these systems have not been pure P2P implementations but rather hybrids that have central servers but take advantage of the scalability of peers to do the heavy computational lifting. These systems include Skype (before it was purchased by Microsoft), BitTorrent, and ‘tisu (an independently developed MMORPG).

**2. Skype Architecture Overview**

For this paper, we will evaluate Skype’s architecture circa 2006 as that is when it was a peer to peer system with a more open implementation. Skype has since been purchased by Microsoft and its implementation has been changed and obscured.

Skype stands out from traditional peer to peer networks in that it has two tiers of nodes: regular and ‘supernodes’[6]. Supernodes tend to have a role more analogous to a traditional server while the regular nodes tend to function more like a traditional client. This is still a peer to peer implementation however because the supernodes are still clients that are owned and maintained by the users of the system rather than the organization behind it. Skype is different from a pure P2P implementation by having the regular nodes associate with an individual supernode without associating with any other node [6]. The supernodes have indexes of each other. Suppose a peer wants to reach another peer on Skype. The peer first contacts its associated supernode with the request. The supernode reaches out to the other supernodes to locate the peer. The supernode associated with the desired peer then responds with the IP of that peer. The two peers then attempt to establish a direct connection between each other and conduct their communication.

This architecture addresses the biggest issue with a pure peer to peer design; the look up of clients and the managing of those lookup tables is computationally expensive. By pulling these tables up into the supernodes it extracts their complexity from the base nodes and cuts back dramatically on the computation and network expense of always updating so many nodes. It also means that the search trees to look up connected nodes are dramatically pruned because they only contain the directed supernodes. This makes the lookups much faster than in a pure peer to peer implementation.

This design would be an issue if there was a lot of turnover in the supernodes as the reassignment of peers is still expensive. However the supernodes in practice are chosen because they have low turnover compared to the rest of the peers. In [6] it was shown that typically the population of peer nodes in Skype fluctuated by over 40% a day while supernodes remained steady with less than 25% fluctuation. Skype accomplishes this by measuring an individual peer’s usage and typical connection statistics and then promoting peers to supernode status based on the networks need and on their likelihood of that peer being stable. As the usage decreases supernodes are similarly demoted to keep the spanning tree reasonably pruned.

Skype’s usage profile is interesting. The majority of Skype traffic is occurs during typical working hours, with usage at night dropping to around 40% of the daytime norm [6]. There were also 20% fewer clients on weekends then on typical week days [6]. This usage pattern is dramatically different from a file sharing service where users typically queue up large batches of files that take days or weeks to process [6]. Skype usage is also significantly different from a traditional telephone network. The average Skype call was 12m 53s while the average call on a traditional phone is three minutes [6]. This is most likely because the VoIP services offered at Skype at the time were free while phone calls were paid for. What is interesting here is that while the traditional telephone network is built on dedicated hardware at great cost, Skype runs on spare network and computing resources.

**3. VoIP and File Transfer in Skype**

VoIP requires decent overall throughput. While the required average throughput is relatively low compared to streaming video or a bulk file transfer, the connection for real time communications has to have a low latency and be consistent.

When direct connections between peers that satisfy these requirements cannot be established the supernodes step in and work to act as proxies and forward traffic between peers. This occurrence is relatively rare with supernodes only relaying data 9.6% of the time with a median network connection rate of 60kbps [6] per supernode. This is because Skype is really good at establishing direct routes between peers as this is often the optimal solution for real time communications. This low usage is despite the fact that Skype maintains a constant rate of transmission for all calls. Skype broadcasts at 33 packets per second even if that whole second was nothing but silence [6]. Clearly this network traffic could be compressed and eliminated to reduce the load on the system even further, but the current implementation is efficient enough that it has not been worth the development time thus far.

Skype also allows for file transfers between peers. These transfers are different from other peer to peer networks because the files are only transferred from one peer to another, instead of from one peer to thousands. This makes them much more akin to a traditional email. These files have a median size of 346kB which is on the order of a document or presentation or photo and is much smaller than a typical file found on a file-sharing network [6].

**5. BitTorrent**

BitTorrent is a P2P file sharing system that uses centralized ‘trackers’ to find files. When a peer wishes to share a file, the peer first splits the file into chunks of data 256kB in size and then creates a meta file containing key information about whatever it is sharing. This file is uploaded to a tracker site and contains the information needed to contact this first seeding peer. Others then use traditional internet search mechanisms to locate the tracker site and get in contact with the seeding peer. The tracker site gives each new peer a random list of other peers who also wish to download the same file. These peers establish among themselves which chunks each of them have, and which ones they need and proceed to share these chunks with each other until they have the completed file. The sharing uses the tit-for-tat mechanism. This means that each peer only gives to another if it receives something in return. This mechanism prevents free loading that was such a burden on the first peer to peer networks.

This has a couple of effects on the network that result from this competition that creates an economy of need for the file chunks.

First peers with high link capacities are favored over those with low capacities because the peers with high capacities can provide more chunks and participate in more in the tit-for-tat dance. If peer A can provide better service to peer B than peer C can, peer B is going to favor peer A and do more business with it.

Second, the connections between peers is symmetrical because peers are often uploading as much as they are downloading. This stands in stark contrast to a traditional design where almost all of the data is from the server to the client. This also makes peer to peer file sharing difficult for ISPs as many ISPs have optimized their services to provide much faster downlinks than up. Since BitTorrent has a symmetrical connection many ISP networks struggle with their atypical load [1].

**6. BitTorrent and TCP**

As the number of peers in a torrent grow the number of requests for various chunks at each peer grows. These requests are queued up at each peer in order to maximize the performance of TCP. TCP uses an exponential startup with rapid back off mechanism. This starts with TCP sending a small number of packets at a time and then waiting for an acknowledgement from the other host. When the acknowledge message is received, the first host responds with putting more packets than it previously attempted into the network, opening the ‘window’ of inflight packets. This continues until the acknowledgement is delayed. Then the window of packets sent at once collapses and the process starts over again. The collapse greatly increases the time needed to transmit data because the bulk of the data is transmitted when the window is further open.

This mechanism is great when there are only a few connections on a client as TCP responds very quickly to congestion. It even works well when there are multiple clients sharing a congested connection because the first client to detect congestion will back off and allow the other clients to continue to keep their windows open for longer and reset their windows gradually, maximizing overall throughput. This protocol is less effective at a single peer on a connection trying to communicate and serve several others all at once. Because several connections on the same peer have a high likelihood of collapsing all at the same time the available bandwidth following a collapse will be wasted while the windows ramp back up to their previous speeds.

**7. Choking in BitTorrent**

Because of this incompatibility with TCP BitTorrent will ‘choke’ connections to try to avoid window collapse. That is it will temporarily stifle one of a peers connections to keep the remaining connection windows open. This will maximize the total throughput for the aggregate connections.

The choking algorithm is very important and it should accomplish four main tasks. First it should avoid sending over too many connections at once to avoid the above TCP problem. Second it should use make sure to implement a tit-for-tat system to serve peers that are serving it. Third it should avoid rapidly choking and unchoking quickly (known as fibrillation) since that will lead to inconsistent performance with the network as a whole. Finally the algorithm should explore new connections to see if they offer superior performance to the current ones.

This last piece is a real life manifestation of the multi-armed bandit problem from the realm of artificial intelligence. The mulit-armed bandit supposes that one is visiting a casino and faced with multiple slot machines (connections to different peers). Each machine has a different payout function (throughput). The challenge is to be able to play each machine such that the payout function can be reasonably determined while maximizing the total profit [7].

One solution is to use an approach called simulated annealing where in a machine is chosen at random and tested for a time. The next machine is chosen at random and obviously always accepted if its available payoff is greater than the current machine. But should also accepted even if it has a worse payout with a probability based on the differences in payout and the Boltsman Distribution. At each step the probability that a worse machine will be accepted is reduced until the probability reaches zero. Simulated Annealing will converge at an optimal solution so long as the time table for reducing the probability is sufficiently slow.

For this real life application in BitTorrent it is not necessary to find the absolute optimal solution so long as one that is ‘good enough’ is found; there typically is not sufficient time for a simulated annealing approach to find an optimal solution for a given download anyway. But the concepts from it can be applied with similar results: select connections at random and test them. Test a more at the start of a download and less at the end. Select the best connections more often near the end of the download and performance will improve for the aggregate. It should now be apparent that the choking algorithm can be very complex.

**8. MMORPG Network Requirements**

Peer to peer networks can also be used for online gaming, particularly when the game is being developed by an independent developer with budget constraints that prohibit deploying hundreds or thousands of servers, particularly when subscriptions can fluctuate wildly. As an example for how gamer populations can fluctuate, at its launch in 2009 Age of Conan boasted having over one million subscribers [3]. However due to the game having many bugs that number was reduced by three quarters in a little over three months [3]. The real problem with client-server architecture (which was used by this game) is that it requires a significant investment in hardware and network infrastructure which is hard to scale. If the subscribers cancel their subscriptions, the hardware has still been invested in and the network connection is likely under contract. These resources are wasted.

On the other side, what if the game had been a hit? If subscriptions had jumped from one million subscribers to one and a half million their existing infrastructure would have been overloaded and sluggish at best. Sure more money could have been invested in more machines but those time to set up, cut into profits and in the meantime the poor performance means that customers are leaving. These challenges face all developers, whether they are backed by a publisher or not. Independent firms have to face these challenges on an even more limited budget.

It makes sense given the elasticity of MMORPG populations to take advantage of the abilities offered by peer to peer networks. MMORPGs have to rely on servers (be they peer to peer or a traditional server) for a few reasons. The first is that clients are not to be trusted to follow the rules of the games, and the servers are used to make sure that all the rules of the game are followed. Second is to handle congested areas that require more computational power to service hundreds of connected clients.

A server for an MMORPG is typically assigned an area of the game world that is their responsibility. These areas are ether discrete chunks, like individual rooms or areas that have set entrances and exits or they are a partitioned quadrant of a larger area. It is much easier to design a system with servers that are assigned noncontiguous areas since users entering these areas will typically go through a door or some other loading screen that will allow the developers to transfer the players onto the server. Even though it is easier to implement, this approach does not scale well if the number of users of an area grows beyond the capacity of the server. If this happens user’s ether need to be denied access which frustrates them or the area needs to be partitioned to be shared by multiple servers [3].

In an area where multiple servers have to share the load developers typically strive to ensure that the boundaries between servers are seamless. This requires that servers that share boundaries need to have areas of each other’s responsibility designated as buffer zones and be in constant contact about what is going on in those zones. This allows users that are on server A but near the border with server B to react to what is happening in the nearby area of server B and vice versa. If users cross a border, they need to have their connections and states transferred to the new server as seamlessly as possible. This communication regarding the happenings near a border has a very high overhead in terms of computational cost and network overhead as objects need to be serialized, sent through the network and deserialized at the other end, all within a matter of milliseconds to avoid disrupting the users experience [3].

Optimizing transfers like this can be hard enough to manage between just two servers, but depending on how a space is partitioned, communication could be occurring between four or more servers because users are near a point where many boundaries meet. Often it is best to partition areas of responsibility into hexagonal cells because then at most a corner of a cell has two other adjacent areas, which limits the number of communications between servers. Using a hexagonal design however has the drawback of more complex computations to determine where boundaries are, compared to a simple grid.

While building a multi-server space is more difficult than having discrete locales, this approach can scale much better to increased users. If the number of users in an area of responsibility is high, divide the space into two or more areas and add another server. All of this discussion is irrespective of the underlying architecture of the MMORPG.

**9. Peer to Peer MMORPGs**

Whether it is client-server or peer to peer these are requirements have to be addressed. As discussed in [3], a pure peer to peer implementation for an MMORPG is a non-starter for two reasons. First a peer should not be in control of its own data to prevent users from cheating and gaining unfair advantages. Second if a peer is not in control of its data, then another peer must be, which is bad if that other peer disconnects and does not return, so the first peer’s data is gone.

A hybrid solution with a limited central server architecture is proposed in [3] and seems like it would be able to address all of these problems quite readily. First there would need to be a central server that would serve two main purposes: a central data store to keep backups of all players’ data, and a load balancer to appoint ‘zone master’ peers. Zone master peers are akin to the super nodes used by Skype: they are peers with stable connections that use spare computational resources to host the service.

To avoid cheating and exploitation, zone masters should not be master of an area in which they are also a peer. They also should be constantly backing up their state to the central server in case they disconnect. It’s up to the developer to ensure that these status updates are fairly lightweight in nature to avoid high computational or hardware costs. Otherwise the zone masters implement all the functionality that a server for an MMORPG area would normally implement including the transfer of players between adjacent areas and acting as a referee to ensure that all players in an area are abiding by the rules of the game.

It makes sense for this kind of implementation to use a seamless world with shifting boundaries since that design can be partitioned any number of ways as load fluctuates. If a zone master’s client starts approaching the area they are in control of the central server steps in and transfers the responsibility to a new zone master. This role of load balancer needs to take into account where various potential and current zone masters are in the game world, where they are likely headed, and their connection rates to the peers in their potential areas of responsibility as well as the zone masters of adjacent areas. It should also factor in the relatively cost of reassigning an area to avoid thrashing control of areas between zone masters.

All communication between zone masters and the central server should be secured using public key encryption to avoid man-in-the-middle attacks and to prevent zone masters from determining which area they have control over.

This solution of using a hybrid architecture was implemented in [3] and found to be highly elastic and fault tolerant. If many new peers are added, then many new candidates for zone masters have been as well, simply nominate however many are needed. If a zone master disconnects suddenly, nominate a new one with the most recent status for that area. This kind of implementation avoids the risks incurred by a traditional client-server approach by limiting the overall hardware needs of the organization to a stable overhead of central servers that grows much slower than a traditional design even if the service itself is experiencing rapid growth or shrinking. This usage will cut costs for development for both large and small firms and facilitate the generation of new games.

**10. Conclusions**

There are several takeaways from this survey of peer to peer applications.

* First, it is growing in popularity and is an effective way to combat elastic needs of computational resources since peer to peer networks are self-scaling. This also helps startups without a lot of capital leverage the computational resources of their users without incurring high costs.
* Second, many of the problems with a pure peer to peer solution (loosing data permanently, finding files) can be addressed with a limited central server backbone as in ‘tisu or BitTorrent.
* Third, there is a common practice of using superpeers as stand ins for much of the functionality of a traditional server as in Skype or ‘tisu.
* Fourth, encryption and security needs to be a prime focus of development. Data is being sent to peers of unknown trustworthiness and that data should be encrypted as it is in Skype and ‘tisu.

Peer to peer networks offer elegant solutions to the growing challenges facing an ever more connected world. Leveraging them in the future will be an important part of getting projects to scale effectively.