Cloud Computing for the Masses

Position Paper

Marc Fouquet
Technische Universität München
fouquet@net.in.tum.de

Heiko Niedermayer
Technische Universität München
niedermayer@net.in.tum.de

Georg Carle
Technische Universität München
carle@net.in.tum.de

ABSTRACT

Cloud Computing provides virtual server infrastructures for companies. The intended benefit is that enterprises do not have to buy their own hardware to provide services for their customers. Therefore, end-users usually do not directly access the raw cloud service (Infrastructure as a Service) as of yet, they use derived services instead.

In this position paper we present a novel way of making cloud-based infrastructure directly useful for end-users by integrating it into peer-to-peer systems. We show one example application which could make use of the cloud without requiring a business relationship between the software vendor and the cloud operator. This software could for example be distributed as open source.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Design, Economics, Human Factors, Management

Keywords

Cloud Computing, Peer-to-Peer, Open Source, Video Streaming

1. INTRODUCTION

Cloud computing has become increasingly popular recently. Companies like Amazon, Google, IBM, Yahoo and Microsoft offer cloud services, mainly for business customers. These services run on virtualized servers in the cloud provider's data centers and are accessed over the Internet. Cloud services are attractive for many companies, as they do not have to buy and maintain their own server infrastructure.

The contracts and payment models of cloud services are the major differences of cloud computing to the classic rented servers. The user of cloud services usually pays depending on used resources, i.e. CPU-hours, data volume or storage. Resources are consumed dynamically according to the current need. The use of virtualization abstracts from the hardware and its problems, which is often considered another goal of cloud computing, as its users do not have to do maintenance of physical servers.

Cloud computing is not one single concept, it is an abstract term. Different cloud providers provide different services. To understand cloud computing, we have to differentiate between three kinds of cloud services that currently exist:

- *Infrastructure as a Service (IaaS)*, provides low-level services like virtual machines which can be booted with a user-defined harddisk image, i.e. Amazon EC2 [1]. Virtual harddisks that can be accessed from different virtual machines are another example of infrastructure as a service.

- *Platform as a Service (PaaS)*, means that the cloud operator offers an API which can be used by an application developer to develop "number-crunching" applications or web applications with friendly user-interfaces. An example is Google's App Engine [2].

- *Software as a Service (SaaS)*, is the flavour which is also useful for end-users. Examples are web-based office applications like Google Docs or Calendar, but also the upcoming gaming service by Onlive. SaaS is usually build based on own or foreign IaaS and/or PaaS.

Cloud computing has been criticized ([7]) by open source activist Richard Stallman for locking the users in with proprietary software, forcing them to continue using the same vendor's products: "One reason you should not use web applications to do your computing is that you lose control. [...] If you use a proprietary program or somebody else's web server, you're defenceless. You're putty in the hands of whoever developed that software." This criticism is targeted at Software as a Service.

In this paper we will explore alternative models of using cloud computing by making infrastructure cloud services useful for end-customers.

In Section 2 we will discuss an example application using our new paradigm. Further use-cases will be explored in Section 3. We will discuss related work in Section 4 and conclude the paper in Section 5.

U-NET'09, December 1, 2009, Rome, Italy.

2. PEER-TO-PEER VIDEO STREAMING USING THE CLOUD

In this section we describe our example application, which basically realizes application-layer multicast for video streaming. The idea is to enable normal end-users to provide streaming video on the Internet like having their own home TV station. We assume the client software to be open source as this makes it easier to show how the interaction between cloud provider and customer works without a commercial vendor of the software.

The bandwidth demand of video streaming is high and peer-to-peer video streaming needs enough upstream bandwidth on the peer's Internet connections in order to succeed. Video streaming can, however, adapt to bandwidth constraints. There are systems which divide the video stream into substreams where a user does not need all substreams to view the video [15]. Simply put, the more streams the user receives, the better the video quality. Advanced approaches build multiple distribution trees for the various streams or, in case of mesh-based streaming[13], operate BitTorrent-like.

In this chapter, we focus on video streaming with one distribution tree, but subsequently show that the concept can be extended to multiple trees and mesh-based approaches.

2.1 Basic Operation

Sender S offers a live video stream and multiple clients want to view the video (see Figure 1). The application that distributes the video is an open source P2P streaming system. It is intended to be used with cloud services of one or multiple cloud providers. It should, however, also work without cloud infrastructure on a pure P2P basis. In particular, we think that it should run initially as pure P2P system. We distinguish between clients A, B, C,... being instances of the P2P application and users A, B, C,... who are the actual persons running the clients.

The problem for the overall video transfer is that the upstream bandwidth of the clients is not sufficient for building a complete video distribution tree so that all clients can view the video. Another problem might be the latency of the links to and from the peers, though this mainly affects applications with stricter realtime requirements and not video streaming which can be buffered in advance. Compared with common DSL users, cloud servers should have a better connection, i.e. no significant gap between upload and download bandwidth as well as a lower latency.

Lets assume that building the distribution tree purely P2P still works when clients A, B and C connect to the stream. The sender has enough bandwidth to support two clients and client C can get its data from client B. But as soon as client D connects, the video distribution does not work any more.

The peer-to-peer application signals this fact to the users and asks if one user is willing to sponsor a relay server. User B decides that he is willing to pay a small amount of money (i.e. 1 US-$ per hour) to improve the video quality for all viewers. This cost model is similar to games in a pub, e.g. billiard or table soccer — one person pays for the table so a whole group can play.

User B has an account at some cloud operator (i.e. Amazon.com) and the services of this operator are supported by the application. In terms of user interface, the payment-related issues at Amazon Web Services are currently not harder than buying books (the same account information is used). However, when it comes to using the services, people are confronted with issues like X.509 certificate generation that are targeted at developers and are not suitable for the average user. So the user interface of cloud operators would have to be changed to make the envisioned system work.

User B enters information about his cloud account into the P2P streaming application. The application does not directly need to know payment information (i.e. credit card information), but it needs to know the user's credentials (i.e. username and password) to obtain permission to spawn virtual machines on behalf of the user. This surely is a trust issue. Even though a malicious application could not directly empty the user's bank account, it could start botnet-style activities like sending SPAM or launching DoS attacks — and the user would be billed i.e. for the traffic. Therefore he has to place a certain trust into the application developer. As a partial solution to this problem, could operators might also offer pre-paid services, so the customer would buy one hour of server usage and get one-time credentials for this.

The next step is the instantiation of the virtual machine. A suitable VM image could have been created by the application developer and placed at the cloud provider (i.e. Amazon EC2 supports public image files). The application instantiates the VM using a web-service call while identifying itself as user B by supplying the appropriate credentials.

After that the distribution tree has to be modified to include the new server. As it is common with the peer-to-peer paradigm, we want to locate the application logic in the end systems. The server itself is only a relatively stupid helper entity which does not participate in the control network by itself. Instead it receives commands from the application on user B's machine — as user B pays for this server, he has direct control over it.

2.2 Building Distribution Trees

There are a number of issues to consider when building the multicast tree. It makes sense to place the strong distribution servers close to the root of the tree and rather have clients as leaves. On the other hand one wants to consider topology information when building up the tree, because otherwise the data might cross intercontinental links unnecessarily often (compare [8]). In fact this aspect could in a limited way be considered when spawning relay servers, i.e. with Amazon EC2 the user can chose between locations in Europe or USA.

Many peer to peer multicast applications, i.e. [14] and [15] are based on multiple distribution trees. The rationale behind this is that peers usually have limited and possibly asymmetric bandwidth and therefore can not relay the full video stream to multiple other peers. Therefore the stream is split into several smaller streams that are distributed using multiple multicast trees. Each individual node is an inner node in only some of those trees and a leaf in the others. When building an actual application from the idea presented in this position paper, following such an approach appears reasonable. The "strong" distribution servers might be inner nodes in multiple trees, and as they are not clients they will be leaves in no tree.

We will now consider some specific details on the distribution tree shown in Figure 2, for simplicity we do this based on the assumption that there is only one distribution tree. Here peers are shown as rectangles while relay servers are circles. One can see that the inner nodes of the tree are mainly distri-

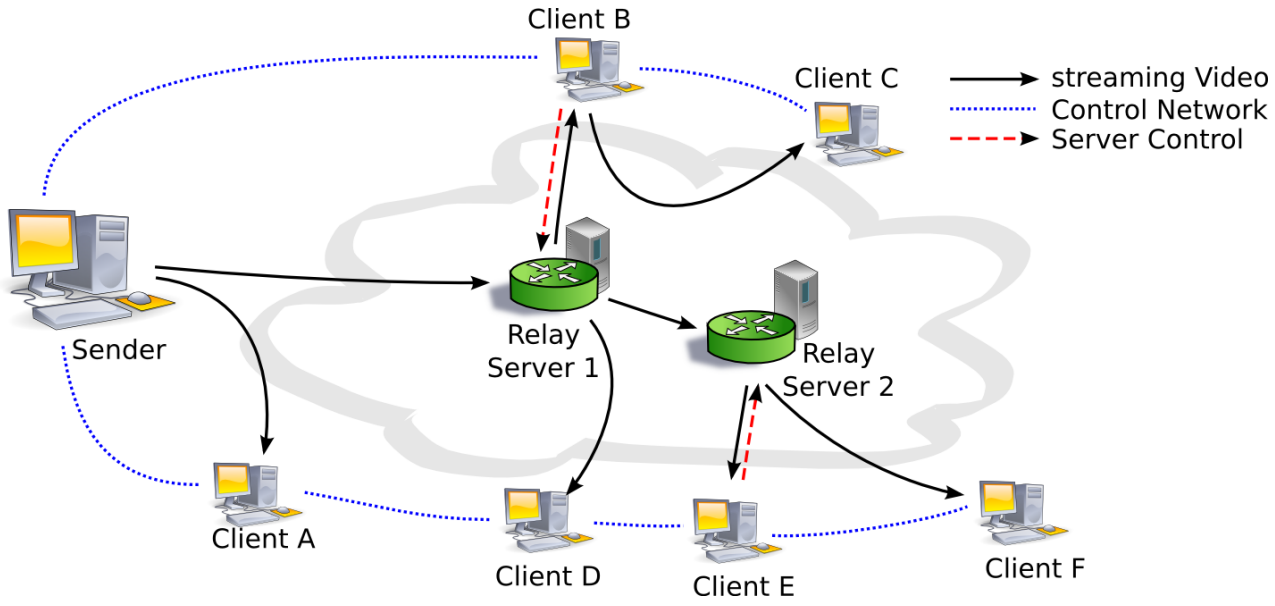

Figure 1: Example application

bution servers, some strong peers also distribute the stream. Peers who sponsor a relay directly receive the stream from their server, so they are often closer to the source than regular peers. The actual leaves of the tree are the freeriders.

There are two situations in which the system might become overloaded:

- *Join of a peer*
  A new peer joins, but there is no more capacity available with the inner nodes of the distribution tree. In this situation

  1. The joining peer is asked if he wants to sponsor a relay server to make the service possible for himself and others.
  2. If this is not the case, all other leaf nodes are asked to sponsor a relay server.
  3. If nobody is willing to sponsor a new server, the new peer will not get a slot anywhere and therefore is unable to watch the stream.

- *Leave of an inner node / relay server*
  A strong peer or a relay server leaves the network.

  1. If the leaving node is not on the lowest layer of the distribution tree (directly above the leaves), one of the lower relay nodes is selected to replace it. This makes sure that the number of affected viewers is minimized — and also that the stream is available for all users who relay. Therefore we may assume without loss of generality that the leaving node was one of the lowest distribution servers.
  2. All of the peers who have lost their relay are asked to sponsor a server.
  3. If this is not sufficient, all other leaf nodes are asked to sponsor a relay server.
  4. If nobody is willing to sponsor a new server, the users who lost their relay are unable to watch the stream.

In practice the actual rules would of course be more complicated. Nodes have different Internet connections with different upstream and downstream bandwidths. As a consequence, we might want to select the relay with the highest capacity as replacement for the leaving higher-layer relay. Still, its capacity may be lower and also some of its peers might have to be discarded.

Some video distributors might prefer a degraded service for the freeriders over kicking some freeriders out if not enough resources are available. In that case, the freeriding peers may share the resources at their relay and get a reduced rate and lower video quality instead.

As seen above, the consequences of a leaving relay server might be severe. Therefore it would be advantageous if the churn-rate of the servers was low. This aspect is considered again in Chapter 3.

2.3 Multiple Trees and Mesh-Based Streaming

Multiple trees can be seen as multiple single tree cases in parallel. Thus, basically the concept can be transferred to that case. For a better operation the rules presented in the previous section have to be reconsidered. If relay servers cannot operate in multiple trees, it might not always be possible to guarantee that relaying nodes always get a good service. However, one can see that the availability of relay servers makes the problem of video distribution easier compared to a pure peer-to-peer approach.

The concept can also be used with mesh-based streaming. In mesh networks the nodes form a random graph and each neighbour is either parent or child. The coordination of the download of chunks is similar to BitTorrent, but with adaptations for realtime transfer. The advantage of these approaches is that clients can adapt to available bandwidth more fine-grained than by joining multiple distri-

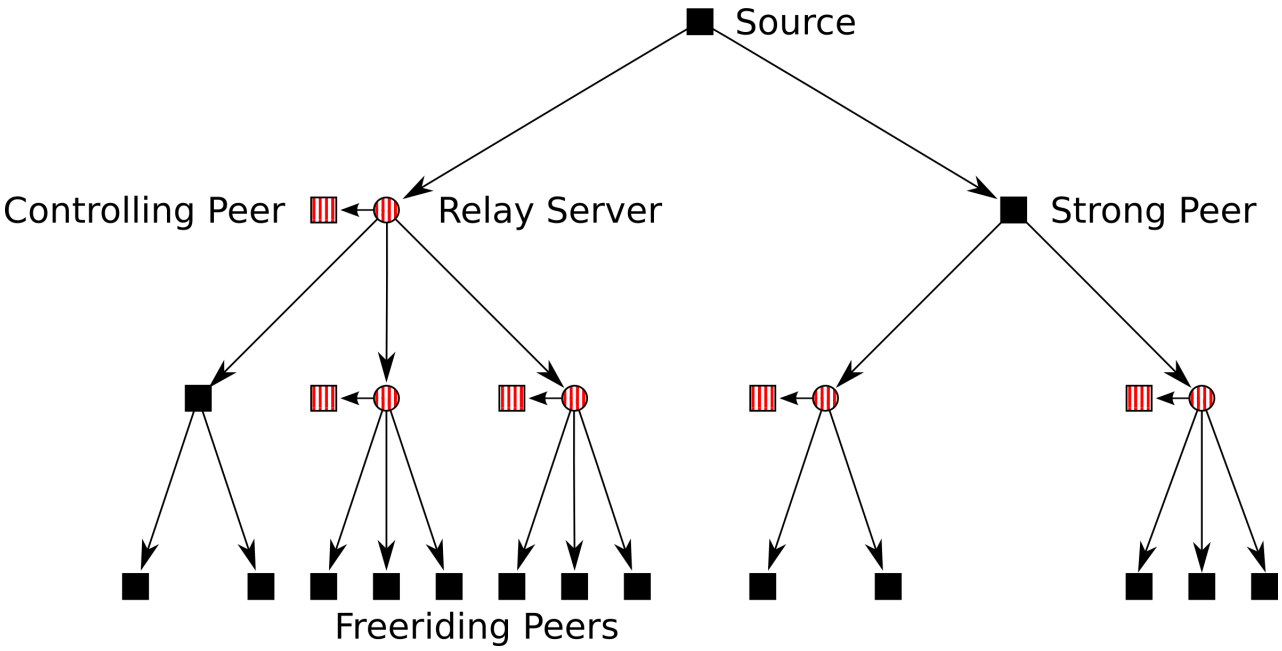

Figure 2: Example distribution tree. Rectangles are peers while circles are relay servers.

bution trees. Each distribution server adds to the available upstream bandwidth while it only consumes the download bandwidth of one stream. Our concept will use the additional bandwidth, so the server increases the overall service quality and number of users that can be served. Users who pay for a server are a child of their server and get their chunks with highest priority, other peers are served with lower priority. For the servers it is beneficial to connect to other servers (as parent or child) and, if still possible, to the source (as child). The broader distribution close to the source reduces latency and increases robustness as new chunks become available at many nodes within a short amount of time.

2.4 User Acceptance and Incentives

The application as described above uses a payment model which is rather uncommon on today's Internet: Some users pay money and sponsor the service for the others. It is not clear whether is model will be understood and supported by the users.

Users who sponsor a cloud server could get certain advantages, like a good position in the distribution tree which guarantees a good video quality. As the server serves them with higher priority and will not discard their packets when resources are out, there is a clear benefit. There are, however, some threats to this.

The advantage for a paying user may not be visible enough to motivate the users to spend money. In our example application the benefits are guaranteed access to the video stream and a high position in the distribution tree which might result in a better video quality. Whether the sytstem works will depend on the target audience and also the explanation of the concept to this audience. There is a risk of too selfish user behavior which might cause a "tragedy of the commons"-situation and therefore a collapse of the system due to the constrained bandwidth.

On the other hand, one could design the system to deliver only a minimal service (i.e. low-quality video) in the beginning. The premium service would initialy be offered to sponsoring users and possibly to a small group of freeriders if the resource situation permits it.

Furthermore, attacks may undermine the concept. Due to the distributed nature of the application, users have various opportunities to cheat for gaining unfair advantages — like claiming that they sponsor a strong server when the server is actually very weak or claiming to be a distribution server themselves. Also a user could sponsor a server, but allow access only to a limited audience and not to the public. Users might sponsor a server to get a good position in the distribution tree, but then pretend that the server is overloaded, so the system spawns other servers on the same level which might reduce the traffic costs for the sponsoring user. There might even be users who actively sabotage the video distribution by inserting malicious relay servers. Such behaviour could severely degrade the service quality, so when actually building such application these aspects should be considered.

Security is out-of-scope for this paper. However, we do think that there are means to mitigate the security problems mentioned above. Cryptographic keys or certificates might protect the clients from accepting a fake VM as server in the distribution network. Those techniques could assure that each distribution server shows the behaviour that was intended by the application developer. However such techniques imply that the users do not have complete control over the relay servers — i.e. they cannot modify the relay software. The server image is created by the application developer and the users may only access it over a well-defined interface. For completely open software projects, this is certainly an open issue.

Home videos as in the example may be a target of low value and lack a business model for certain attacks. For

larger video distribution projects this assertion certainly does not hold.

3. FURTHER USAGE SCENARIOS

It is not new for peer-to-peer networks to rely on a set of more central components. Many peer-to-peer systems are based on so-called "supernodes", peers that can perform special tasks because they have certain properties, like stronger CPUs, more network bandwidth or persistent and direct (non-NAT) Internet connections.

Cloud servers can be used to add such powerful nodes in a controlled way if needed. The advantage is that the higher fraction of strong nodes in the network enables applications that would otherwise not be possible — like the distribution of high-quality live video.

One issue that these nodes could address — possibly with support of the cloud operators — is churn. Churn describes the rate at which peers join and leave the network. A high churn rates means that a lot of peers come and go. This is a problem for almost any network, as churn degrades its performance and requires appropriate maintenance overhead.

Currently, Amazon EC2 servers are paid per hour. If one would modify the proposed system in a way that assures that the cloud server will stay online for the rest of the already paid hour, even if the client disconnects, the remaining online-time of these strong nodes would be predictable.

We think that a large number of applications — not all of them peer-to-peer — could benefit from cloud servers that are directly controlled by the users:

- *Data distribution applications*, i.e. filesharing or distributed file storage.
- *Privacy enhancement applications*, i.e. applications that perform onion routing.
- *Circumvention of censorshop* in countries which block free Internet access.
- *QoS improvement* by "source routing" around congested areas or network failures (cf. [9]).
- *Circumvention of Network Adress Translation*, i.e. if the NAT is not under the User's direct control.
- *Private Networks*, i.e. building a VPN or tunneling IPv6.
- *External Firewall*, i.e. for filtering traffic before the bottleneck (last mile) link in case of DoS attacks. This could i.e. be useful against small botnet attacks that recently occurred against players of online games on Microsoft's XBox Live Service.
- *Building large-scale social peer-to-peer applications*, i.e. a serverless and non-commercial social network which gives the users more control over their private data.

Today's use-cases of Cloud Computing are all more or less commercial in their nature. Our idea bridges the gap between commercial and non-commercial use-cases with the distributed voluntary payment model. As a consequence, we think that this idea makes cloud computing applicable in scenarios that have not been possible before.

- *Open Source Software* is almost impossible to provide as SaaS as the philosophies of classical SaaS and free software are fundamentally incompatible. With our model open source software can use benefits of cloud computing, i.e. running in a standardized environment on virtualized hardware and having extremely fast internet connections.
- *Any distributed application whith an increased need for transparency and control by the user*, i.e. privacy enhancing applications. This also applies to businessess who would not want to store confidential documents on network storage which is supplied by an external provider.

Further the model allows to implement features like global multicast that have been considered desirable for a long time, but are not supported by ISPs because of security and charging issues. Our model makes individual users accountable for replicated traffic and therefore solves these issues.

Using cloud computing the way we propose does not only open it to new use-cases. Additional business models seem to be attractive as well. Peer-to-peer software developers or video providers still lack means to get a revenue from their service. In the motivational example they do not have the means to accept and enforce payments. If they had business relations (or simply accounts) with cloud providers, they could share the revenue and determine appropriate prices.

4. RELATED WORK

Research on peer-to-peer video distribution has been going on for years ([15], [10],[13]). At Stanford University, a system was designed which allows timeshift and is currently being extended to support mobile devices [14]. Limited upstream bandwidth of peers generally is a problem with peer-to-peer video distribution [12]. Therefore some approaches try to combine peer-to-peer video distribution with server-based content distribution networks, i.e. the authors of [11]. Such hybrid approaches are commercially exploited by companies like Octoshape [5] and Joost [3].

The idea to use cloud services for peer-to-peer networks is related to the concept of super peers [16] that has been adopted by Skype [6] as well as in filesharing networks like Gnutella2. While it is a challenging task to find appropriate peers for being a super peer in usual peer-to-peer networks, in our concept the cloud servers as super peers can be spawned as desired. In contrast to our example application's relay servers, traditional super peers have more functionality and are usually responsible for the management and maintenance of the network.

Cloud computing services are available from a number of companies, for example Google [2], Microsoft [4] and Amazon [1]. Especially Amazon Web Services are a perfect candidate for realizing our system, all that would be required are changes to the user-interface to make the Elastic Compute Cloud (EC2) service usable for non-developers.

5. CONCLUSION

In this work we presented the idea of putting cloud computing services under direct control of their users or applications on the users machines. Such an approach nicely supplements existing peer-to-peer systems by supplying the networks with urgently needed resources, i.e. for live video

streaming applications. This makes it possible for users to cooperatively deliver services that would otherwise require a large amount of infrastructure.

We have pointed out a number of usage scenarios for this paradigm and discussed the video distribution example in more detail. Technically the proposed systems are already possible today (i.e. using Amazon EC2), only the user interface of the cloud operators needs some improvements.

6. REFERENCES

[1] Amazon Web Services. http://aws.amazon.com/.

[2] Google App Engine. http://code.google.com/intl/en/appengine/.

[3] Joost. http://www.joost.com/.

[4] Microsoft Azure. http://www.microsoft.com/azure/.

[5] Octoshape. http://www.octoshape.com/.

[6] Skype. http://www.skype.com.

[7] The Guardian: Cloud computing is a trap, warns GNU founder Richard Stallman. http://www.guardian.co.uk/technology/2008/sep/29/cloud.computing.richard.stallman.

[8] V. Aggarwal, A. Feldmann, and C. Scheideler. Can ISPs and P2P systems co-operate for improved performance? *ACM SIGCOMM Computer Communications Review (CCR)*, 37(3):29–40, July 2007.

[9] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. *Computer Communication Review*, 32(1):66, 2002.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth content distribution in cooperative environments. In *Proc. of IPTPS'03, Berkeley, CA*, 2003.

[11] Y. Dong, E. Kusinierek, and Z. Duan. Exploiting Limited Upstream Bandwidth in Peer-to-Peer Streaming. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, pages 1230–1233, July 2005.

[12] W. Gao and L. Huo. *Challenges on Peer-to-Peer Live Media Streaming* , pages 37–41. Number LNCS 4577. Springer, 2007.

[13] N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-drIven MEsh-Based Streaming. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1415–1423, 2007.

[14] J. Noh, P. Baccichet, F. Hartung, A. Mavlankar, and B. Girod. Stanford Peer-to-Peer Multicast (SPPM) – Overview and Recent Extensions. In *Proc. International Picture Coding Symposium, PCS 2009, Chicago, IL (Invited Paper)*, 2009.

[15] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. In *Proc. of ACM NOSSDAV, Miami Beach, FL*, pages 177–186, 2002.

[16] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. *International Conference on Data Engineering*, 0:49, 2003.