Introducing Octoshape

— a new technology for large-scale streaming over the Internet

Stephen Alstrup and Theis Rauhe
Octoshape

The popularity of live streaming over the Internet is growing. The number of private Internet connections are rapidly expanding and consumers may today go online from almost any location using wireless technologies. However, live streaming over the Internet was born with two problems: *scale* and *cost.* The challenge is to transmit a signal to many users simultaneously and to do so without the transmission cost rising in proportion to the audience size.

In this article, we will walk through various technologies for transmitting live streaming, including the more traditional ones, to see how the problems of scale and cost occur. Then, we will look at how some of these technologies partially solve these problems. Finally, we will present and explain *GridCasting* – which offers a solution to both problems – and then look at how Octoshape takes advantage of it.

Live streaming over the Internet raises many technical questions [1][2][3], to which you can get a general introduction in Franc Kozamernik's article *Webcasting – the broadcasters' perspective* [4]. This article focuses on the ability to transport live streams in large scale and without costly Internet resources.

By live streaming we mean real-time transportation of live or stored media over the Internet. Unlike with downloading or VoD (Video on Demand), with live streaming you can transmit an event as it actually occurs. (For a detailed overview of the many concepts and techniques used in delivering streams, see Franc Kozamernik's article *Media Streaming over the Internet – an overview of delivery technologies* [5].)

Scale and cost problems are the spoilers

We have already mentioned that there are problems associated with scale and cost. These problems arise because the Internet and its mechanism were designed to transfer files from point-to-point, where the receiver accesses the file after it has been fully downloaded.

This is quite contrary to the idea of transmitting theoretically endless streams from one source to many receivers. For example, a broadcaster with thousands of simultaneous listeners of an audio signal also needs to send thousands of separate streams – one to each listener. This is why the bandwidth (Internet capacity) needed to stream over the Internet and thereby the cost are proportional to the size of the audience.

Furthermore, it also explains why it is hard to scale. Sending a separate stream to each member of the audience causes bottlenecks on the Internet. Bottlenecks imply that the signal is unstable for the end user and, in the worst case, the user receives a "Server is busy" message.

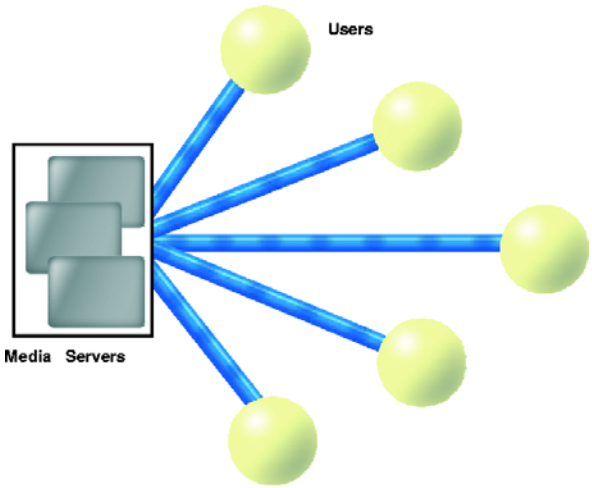
Nevertheless, the demand for more and better quality live streaming over the Internet is rapidly increasing. The scale and cost problems, however, make it difficult to capitalize on the popularity of this new media.

Solutions for live streaming on a large scale

Several approaches have offered a partial solution to the scale and cost problems. In this article, we will review these approaches and then go on to present and compare them with ***GridCasting*** – a technology offering a solution to both problems.

Up until now, grid technology has been used for tasks such as file-sharing with Peer-to-Peer (P2P) programs like BitTorrent to distribute movie and music files. Users download from each other without costly central sources. In the last part of this article, we will discuss how GridCasting may be applied to live streaming and offer the same benefit: large scale broadcasting at very little cost. Since GridCasting is a pure software solution, it will also be possible for broadcasters to have their own distribution net.

Media Server Farm: unicasting and balancing the load



Unicasting is a classic approach to live streaming. Here, the setup is to use one or more clustered servers (basically expensive PCs) to manage the request from users to receive a stream. Clustering balances the load on the machines and makes the setup robust even if one of the servers breaks down. The servers in the cluster share an Internet line through which the streams may be sent to the users. Such a setup is called a ***Media Server Farm***. The size/capacity of an Internet line is denoted by the bandwidth.

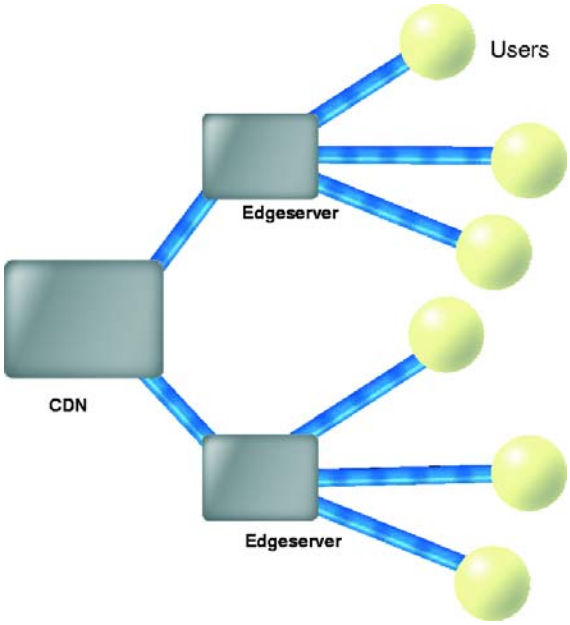
If, for example, the shared Internet line from the media server is 100 Mbit/s, then an audio stream at 100 kbit/s may at most be sent to 1000 simultaneous listeners. Some large broadcasters have two or three Media Server Farms (one in the US and one in the EU, for example). Nevertheless, the bandwidth needed is still proportional to the size of audience. This can be represented as follows:

```
Bandwidth_Needed = Size_of_Audience X Streaming_Quality
```

The cost problem: Doubling the quality requires double bandwidth; tripling the audience requires tripling the bandwidth. In other words, as the quality of the broadcast and the size of the audience increases, so too do the costs.

The scaling problem: Since all the streams are sent to the Internet from one source (or perhaps two), this approach quickly reaches its upper limit. Bottleneck problems and "Server busy" messages occur. The unicasting approach described above may be compared with a traffic situation where all travellers begin their trip to work at the same time in the morning from the same house. Simply put, it is difficult ... and it also explains why doubling the bandwidth capacity can more than double the price – making it a "mission impossible."

CDN: being present worldwide



The Content Delivery Networks (CDN) approach was originally developed to solve the waiting problem on the World Wide Web: handling many simultaneous requests at the same homepage effectively. The main idea was to place thousands of machines, so-called *edgeservers*, around the world at strategic points on the Internet. The same homepage would then be placed at all the edgeservers. Then, when a user requests a homepage, he or she would get it from an edgeserver located nearby. This approach increases the possible numbers of simultaneous requests that may be handled, since the load on the machines and bandwidth needed are distributed. Today, CDN is also used for live streaming and a few enterprises offer this solution.

In the live streaming solution, the approach is similar to the homepage solution. The above description, however, only gives an overview of a CDN solution. (For further details, we refer to one of the few enterprises that offer such a solution, Akamai [6].)

Clearly, the CDN solution is more scalable than a Media Server Farm, having thousands of edge-servers placed worldwide. However, this is only fully true if the audience is distributed similarly to the edgeservers, so that the bulk of the burden is not placed locally.

For example, if you want to broadcast a big event in the UK, it would not help much to have thousands of machines located outside the UK. All the machines outside the UK would have to send through the same lines into the UK anyway. Or, to return to our morning traffic metaphor, it would be the same as trying to send all travellers to work in the UK from starting points outside the UK. So, while CDN solves the scale problem to some extent, it does not solve the cost problem because it introduces thousands of machines that need to be maintained.

Multicast: one stream – many addresses

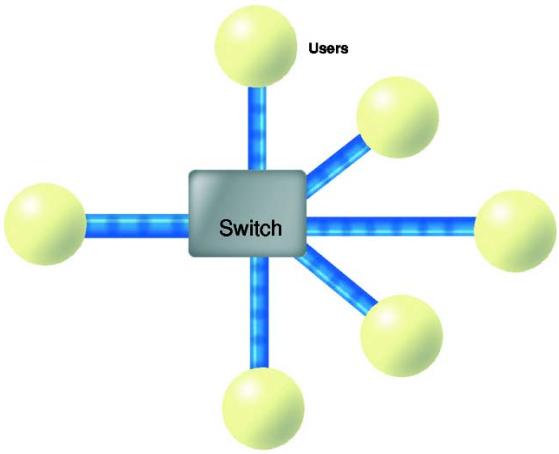
Ask five people for a definition of multicast technology, and you will most likely get at least seven different answers. Here, we offer two definitions, which hopefully will cover the most common views. However, in order to present these definitions, we must first offer a brief definition of an Internet Protocol (IP) number.

IP number

Driving in a car to a specific place, you need a specific address. In the same way, when sending streams over the Internet, you need to have it marked with an address. This address is called the IP number. Without giving details, one could say that each machine hooked up to the Internet has its own unique IP number, which a stream uses to find its way to the machine it is meant to be delivered to.

Classic multicasting

A few special IP numbers are reserved for multicasting, and these are called multicast IP numbers. The main idea is that one may address a stream not to be sent to a specific machine but to a multicast IP number. Any information sent to the IP number will be forwarded to all that have signed up



as listeners. The idea of multicasting has been around for a relatively long time (measured in the fast pace of the Internet era). It dates back to the 1980s [7], and many efforts have been made to make it work on the Internet. For an example, see the BBC's current effort [8].

Although a multicast IP number is not a physical entity, it has to be supported by hardware and software. The big questions are: Who enables it? How is it enabled? What would happen if virtually anyone could use it, and who pays for it? Today, it is possible to multicast within an enterprise, for example, if the switches/routers in the local network/intranet are multicast-enabled.

Hardware multicast

An Internet Service Provider (ISP) who possesses and/or controls a part of the Internet – all the way out from a central point to his customers (DSL subscribers) – may set up a hardware multicast solution. It can be costly for an ISP to multicast-enable their network but when done, the ISP can then send multiple streams to all the customers on this "local network" offering services similar to a cable network.

Last comment about multicast: Even when multicasting works in both of the ways described above, there are still issues to deal with, such as congestion problems. Furthermore, multicast introduces a strict one-way type of communication, which involves other challenges such as how to gather statistics.

P2P live streaming: sharing the burden

When using any of the technologies described up until now, the users receive a stream but do not send anything. However, a user is both capable of sending and receiving. Thus, the above solutions place the full burden of transmitting to the end users – on the broadcaster (or his hosting provider). Meanwhile the end users' sending capacity is unused, standing in an idle mode.

Note: In order to use the idle bandwidth at the end user, the end users must have a shared plug-in installed on their machines. In the remainder of the article, we will sometimes refer to "the end user sending" which more precisely means "the plug-in at the end user is sending. "

With P2P live streaming, the idea is to use some of the idle sending capacity at the end users. This reduces the burden on the broadcaster and may be represented as follows:

Bandwidth needed for the broadcaster =

```
Streaming_Quality X Audience_Size – Used_Idle_End user_Bandwidth
```

A broadcaster could send the stream once to one end user, who would then forward the stream to another user, who again would forward the stream, and so on. In theory, the broadcaster could have an audience of millions of people while only sending the stream once. For a number of technical reasons, however, such an approach – with a long chain of users forwarding the stream to each other – does not work.

A way to avoid the long chains

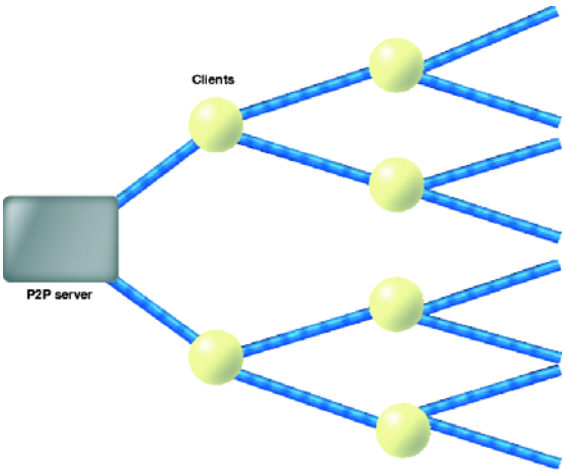
In practice however, another approach – based on the principle of using the end users' sending capacity – does work to some extent. In fact, several commercial software enterprises [9], open source projects [10] and university projects [11] are using this approach every day. With the excep-

tion of a few theoretical research ideas, they are all based on a "tree structure": the broadcaster sends the signal to two users, who forward the stream to two or more users, who again forward the stream to two or more users and so on. In this way, they avoid the very long chains mentioned earlier. This approach is called ***P2P live streaming***.

This construction has been reported [12] to be successful for live streaming on a small scale with groups of 10-100 simultaneous receivers. This may be scaled to some extent by using several groups/servers. Streaming on a small scale and at a low bitrate (low quality) seems to reduce the bandwidth needed by 40-50% compared to the bandwidth needed for a media server farm.

The gains are more or less in theory

In this diagram, it seems like the P2P server only needs to send the stream twice, therefore achieving an almost 100% saving. However, in the following we will discuss some of the issues that explain why the full theoretical gain is not obtained in practice:



- End users need to send double/triple as much as they receive. However, DSL is set up to send from host to end users, not the other way around. One way to compensate for that is to use only low bitrates.
- In a tree structure, it is a mathematical fact that most users will be in the last level, called leaves, which only receive but do not forward any streams. Thus, only a minority of the users will be contributing with idle bandwidth.
- When broadcasting a 150 kbit/s stream, for example, only users that may forward at least 150 kbit/s can contribute. So, a user with a sending capacity of 128 kbit/s will not be able to contribute at all.
- When one user turns off his or her machine, perhaps one near the original source / P2P server, the users he or she has been forwarding to (as well as the ones that they have been forwarding to) will lose their stream and contact the P2P server which will then have to rearrange the tree.

For these and other technical reasons, it is hard to create robust and stable streaming using P2P-servers on a large scale. Moreover, it will only become more difficult in the future. The reason for this is that the communication protocol (TCP) used today for P2P streaming does not allow end users to send to each other if they are behind a NAT (Network Address Translator) which is an inherent part of wireless routers (with the exception of end users who are PC experts and are capable of opening ports on the routers). The concept of TCP and NAT is explained in [5].

Note: One could imagine that extending the CDN approach with a P2P solution [13] would strengthen CDN at its most vulnerable point – the high load placed on a local location.

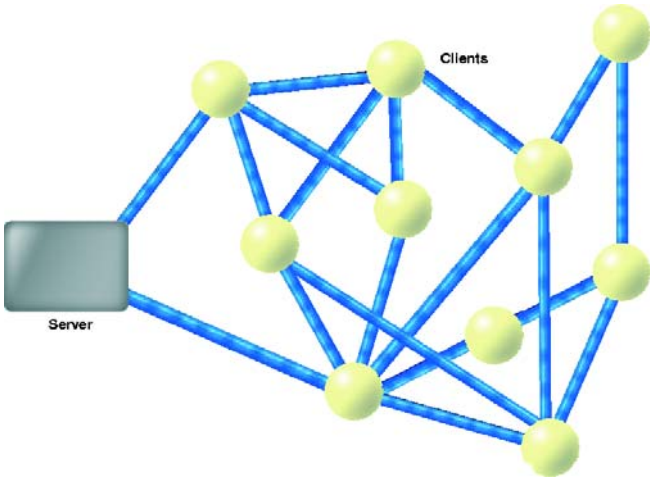
GridCasting: Octoshape's solution for large-scale live streaming

Octoshape's GridCasting solution saves 97% of the bandwidth needed compared to the traditional server farm solution. Since almost no bandwidth is needed, there are no bottleneck problems – making the technology scalable.

As in the P2P "tree structure" approach, Octoshape's solution benefits from end users' idle bandwidth, using a plug-in, to reduce the bandwidth needed for the broadcaster. This, however, is the only similarity with the tree structure, which we discussed earlier. Octoshape looks much more like

the widely used programs for file sharing. So, before going into detail on the Octoshape GridCasting solution, let's take a quick look at file-sharing programs.

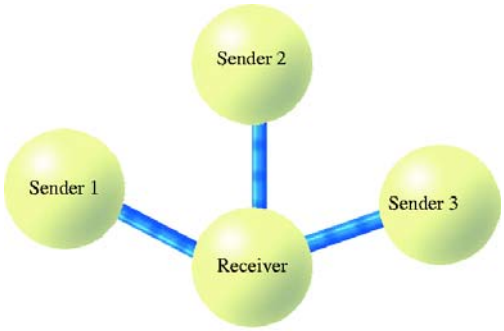
File-sharing programs for downloading



File-sharing programs like BitTorrent [14] and Kazaa [15] are widely used to share media files such as movies and music. End users share files on their PCs by sending to and receiving from each other. In fact, reports (see e.g. [16]) indicate that such file-sharing programs represent approximately 70% of all Internet traffic. The BBC is testing a solution (iMP [17]) that allows end users to share BBC media. In this way, the BBC may limit its bandwidth cost significantly. File-sharing programs cannot be used for live streaming but only to download e.g. a music file. The music cannot be played until the download has been successfully completed.

As illustrated above, the server delivers some data to the users. Afterwards, the users exchange the data among themselves which reduces the bandwidth needed for the broadcaster. Not only may a single user send to several other users ... the single user may also receive data simultaneously from several users. Being able to receive simultaneously from several other users is the definition of a *grid* which is crucial to the efficiency of the system.

In the P2P tree structure, a user only receives from one single user. Let us have a closer look at the single user, known as the "Receiver" in this diagram, to highlight the importance of receiving simultaneously from several others. This user is receiving from three other users called "Senders 1, 2 and 3":



Here, we are still considering the scenario where end users download media files from each other. In the diagram *(right)*, the receiver could be downloading a movie, and three other users could be sending parts of the movie to him, e.g. the first, the middle and last 30 minutes of the movie. Two main features of a grid are:

- You can receive different parts of the data/media file simultaneously from several other grid users. This allows several users to send to the same receiver, and the receiver to use its download capacity to the full extent.
- If the connection from a sender is congested or the sender turns off the PC, the receiver may choose another sender among the end users joining the grid. In this way, no one is dependent on the behavior of one single sender.

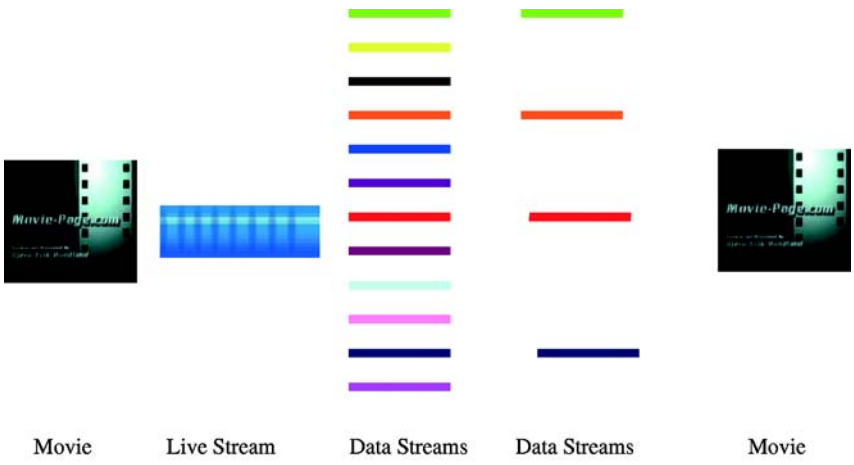
Grid technology for live streaming?

Using a grid for downloading is much simpler than using it for live streaming. If a sender turns off the PC, and time is needed to find another sender, it will only extend the time you need to download. In the live streaming case, however, it could involve that you lose the signal completely until another available user is found. Furthermore, it is not obvious how to apply the grid for live streaming: one cannot receive the first, the middle, and last 5 minutes simultaneously from several senders, since a first, middle and last part do not exist. The only thing that exists is the signal that is transmitted live.

This explains why previous technologies that tried to exploit the users' idle bandwidth used the unreliable tree structure as explained in the section called "P2P live streaming. "

The key to using grid technology for live streaming

Octoshape's Live Streaming technology consists of several components to obtain the full benefit from the grid approach. In the following, we will explain some of the techniques used.



The *movie* (or radio) signal is transmitted as a live stream. From the *live stream,* several *data streams* are constructed. No *data streams* are identical. For the purpose of this example, assume the *live stream* is a 400 kbit/s signal and each of the *data streams* has a size of 100 kbit/s. In the Octoshape solution, a large number of unique *data streams* are constructed. In the example above, 12 *data streams* are pictured. Now, an end user receiving any four of the 12 different *data streams* of 100 kbit/s may use these four *data streams* to construct the original *live stream*, and thus the *movie* can be played in real time at the end user.

The single user's perspective

In the following, we focus on the perspective of a single user in the grid. In the Octoshape live streaming solution, each end user has his or her own *unique data stream*. Thus, the number of unique *data streams* equals the number of end users. These data streams are produced in a distributed way, which does not burden the broadcasters. While the technique used to do this is not presented here, the result will be explained. Each end user sends his/her unique data stream to 0,1,2,3 or more users and collects (in this example) four data streams of 100 kbit/s from any four other users in the grid. He or she can then put them together to construct the original live streaming signal.

In this diagram, the user sends a *data stream* to three other users and receives a *data stream* from four others.

Normally, one would try to minimize overheads such as metadata when streaming in order to reduce the bandwidth used (Overhead is anything besides the audio/video signal). The Octoshape technology, however, turns everything upside down. It assumes that live streaming does not require bandwidth which brings us to the second technique: When live streaming the content (a radio signal, for example), the information about who is also joining the grid is sent as metadata with the stream. In this way, each single user (not the person but the Octoshape plug-in on the user PC) has an address book of other users in the grid.

In the following, we will give examples of how these components achieve the full benefit of the grid approach.

Stable connection: The end users also maintain a standby list of end users. Each end user continually probes/asks the users from his standby list to be sure that they are ready to take over if one of

the current senders stops sending or gets congested. If the standby list is too small, end users from the address book are added.

Avoiding bottleneck problems: If the connection to a sender gets congested, the user will exchange it with one from its standby list. Since all users continually monitor connections involving themselves, bottleneck and congestion problems on the Internet are avoided.

No server load: Since each end user has an address book and a standby list, he/she does not have to burden the server when a sender stops sending or gets congested.

Flexible use of end users' bandwidth: Since users participate with their idle bandwidth by sending data streams of a size smaller than the live stream, a very flexible system is achieved. A user with an upload capacity smaller than the size of the stream will just send fewer data streams, and a user with a higher upstream capacity may send more data streams. To gain the full flexibility, the ratio between data and stream size is bigger than 4 as illustrated in the example above. To achieve a 97% saving, the only demand is that the users have an *average* sending capacity as large as the live stream.

Sending a TV quality signal: If the average end user sending capacity is 300 kbit/s and the live stream is 400 kbit/s, there will be 100 kbit/s less bandwidth per user in the grid. In this case, an artificial end user may be added to deliver the extra bandwidth to the grid.

Doing the impossible: Imagine a group of end users, somehow geographically associated. You might want to think of end users in the UK, in a enterprise or in a city. This group of end users is connected to the rest of the Internet through a few lines. If sending from a source not located geographically within this group, and doing it by using unicasting, then all the traffic would have to go through the few lines connecting the group to the rest of the Internet. When using GridCasting, the signal only has to reach one end user in the group, which then can spread it to the rest.

In conclusion:

All users may contribute to the broadcast;

- with any fraction or multiple of the stream size;
- without loading the server when establishing a connection;
- and without local and global congestion and bottleneck problems.

Therefore, a scalable and 97% bandwidth-saving technology is achieved.

More facts, figures and details

It is beyond the scope of this article to describe all logical questions but we will try to give the answer to some of the most obvious ones.

Saving: The larger the audience, the higher the percentage saved. When streaming to more than 300 end users, the saving will in fact be higher than 97%. On the other hand, streaming to only one person will offer no savings. The reason we say the saving is 97% for large-scale streaming and not 99% is that the burden of logging, initial handshake with the server, and so on, cannot be distributed among the users in the grid.

Abbreviations

CDN	Content Delivery Network	**ISP**	Internet Service Provider
CEO	Chief Executive Officer	**NAT**	Network Address Translator
CTO	Chief Technology Officer	**P2P**	Peer-to-Peer
DSL	Digital Subscriber Line	**PhD**	Doctor of Philosophy
IP	Internet Protocol	**TCP**	Transmission Control Protocol
iMP	(BBC) interactive Media Player	**VoD**	Video-on-Demand

Setup: Currently, Octoshape have several servers set up in the EU and the US in a multiple fallback system. This is because although only a small amount of bandwidth is needed, it should be guaranteed. Furthermore, if the end user does not use Octoshape to receive the stream or cannot use Octoshape (e.g. due to explicit blocking in a firewall), the signal has to be delivered to the end user anyway. This is also the reason why Octoshape recommends a twofold setup. For example, if a broadcaster today has a 20 kbit/s radio signal, the setup will be a 20 kbit/s signal without Octoshape and perhaps a 96 kbit/s signal with Octoshape. It is now up to the end user whether he or she prefers the higher and better quality and, if so, the broadcaster saves bandwidth and thereby costs.

End users: PCs are not user-friendly when it comes to streaming technology. If, for example, one user installs a particular media player, that player sometimes associates itself with a codec that it cannot use. The result is that nothing happens when the end user clicks on the media link. End users are also often asked to make detailed choices regarding players, Internet connection speeds, codecs, browsers and so on.

For these reasons, Octoshape incorporates a linking technology in the plug-in that makes streaming much easier to use. If the user, for example, has installed a player which insists on attempting to play a stream that it cannot play, the Octoshape plug-in detects the situation and simply launches one of the user players that can play the stream. If such a player does not exist, the user is notified of the problem.

Latency: In streaming technology, latency refers to a delay in the playing of the live stream. Since Octoshape is using a much more robust technology, the buffering size, and hence the latency, may in fact be reduced.

Security: To guarantee security in all senses, Octoshape has taken a number of steps including technologies that are well known from Internet banking: the plug-in runs in a secure “sandbox” and there are automatic updates, encryption etc. Furthermore, only communication verified by a central server is allowed. This last feature also makes it possible for the broadcaster to fully control who should and who should not receive the signal.

The bottom line

GridCasting is a new technology which allows live streaming on a large scale without costly central resources. With this new technology, the main barriers to successful use of this new media have been removed.

Stephen Alstrup, CEO Octoshape, is a leading expert in algorithms who has published a number of papers within the areas of data structures and, in particular, tree structures. Prior to founding Octoshape, he was associate professor of The IT University of Copenhagen, where he established and led the Algorithm Group. Dr Alstrup received his PhD in Theoretical Computer Science from the University of Copenhagen in 1999. During his professional carrier, he has worked as a consultant and in research groups on large-scale problems for enterprises such as AT&T, Google and Microsoft.

Theis Rauhe, CTO Octoshape, is a world-renowned algorithm scientist who has made significant contributions in the areas of data structures, lower bounds and software engineering. Prior to founding Octoshape, he was associate professor at The IT University of Copenhagen and a member of various research committees. He has lead several large software projects founded by organizations such as the European Union. Dr Rauhe received his PhD in Theoretical Computer Science from BRICS at the University of Aarhus in 1999.

References

[1] Kari Bulkley: **Broadcasting over the Web**
EBU Technical Review No. 293, January 2003.

[2] Franc Kozamernik and Gerhard Stoll: **EBU listening tests on Internet audio codecs**
EBU Technical Review No. 283, June 2000.

[3] Ivar Poijes: **Streaming audio in the networked environment**
EBU Technical Review No. 296, October 2003.

[4] Franc Kozamernik,: **Webcasting – the broadcasters' perspective**
EBU Technical Review No. 282, March 2000

[5] Franc Kozamernik: **Media Streaming over the Internet – an overview of delivery technologies**
EBU Technical Review No. 292, October 2002.

[6] Akamai: **http://www.akamai.com/**

[7] RFC Editor: **http://www.rfc-editor.org/**

[8] BBC Multicast: **http://support.bbc.co.uk/multicast/**

[9] Allcast: **http://www.allcast.com/**

[10] Peercast: **http://www.peercast.org/**

[11] ESM: **http://esm.cs.cmu.edu/**

[12] Y. Chu, A. Ganjam, T.S.E. Ng, S.G. Rao, K. Sripanidkulchai, J. Zhan and H. Zhang: **Early experience with an Internet broadcast system based on overlay multicast**
Proceedings of Usenix, 2004.

[13] Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs and Hui Zhang: **The Feasibility of supporting Large-Scale live streaming applications with dynamic application end-points**
SIGGCOM, 2004.

[14] B. Cohen, Bittorrent, 2001, **http://www.bitconjurer.org**

[15] Kazaa: **http://www.kazaa.com**

[16] Cachelogic: **http://www.cachelogic.com/index.php**

[17] BBC's iMP solution:
http://www.bbc.co.uk/pressoffice/pressreleases/stories/2005/05_may/16/imp.shtml