Reliable Unicast Streaming with Multiple Peers in IP Networks

Florian Unterkircher and Michael Welzl

University of Innsbruck, Computer Science Dept.
Technikerstr. 25, A-6020 Innsbruck, Austria
florian@unterkircher.com, michael.welzl@uibk.ac.at

Abstract. The traditional client/server model with its single point of failure and centralized administration has turned out to be a poor fit for file downloads for some applications; the same may apply to streaming media services in the near future. We propose an architecture for streaming media in a peer-to-peer network scenario that is robust and efficient, and expected to be advantageous even if links are asymmetric.

1 Introduction

It is widely expected that the live streaming of media content such as digital music and video will form a significant fraction of internet traffic in the near future[1]. Multimedia data streaming in IP networks is currently mostly implemented as a client/server or multicast architecture. A variety of problems has been identified with this traditional approach; we believe the most important ones to be:

1. *Flash crowds:* Established streaming solutions have shown not to hold up well against a flash crowd of users who request multimedia feeds from the same source; this was particularly evident on September 11th, 2001. Architecturally, it is common knowledge that single points of failure and bottlenecks should be avoided. Peer-to-peer based caching and forwarding schemes have been demonstrated that effectivively alleviate the effects of flash crowds [2] [3].
2. *Bandwidth costs:* Bandwidth costs can pose a huge problem to the content provider if he cannot charge for each stream. A prominent example of a business that failed because of high bandwidth costs[1] is AdCritic, once a popular site that offered streaming video versions of the best rated television commercials [4]. The view that streaming of data from a central source is too costly is also supported by CNN's recent decision to charge for access to its video content on their website [5]. It would be beneficial if the bandwidth

[1] During Super Bowl in the year 2000, for example, more than 12 terabytes of data were downloaded from the company's streaming host by 1.8 million users, at an estimated cost of USD 120,000 to the company.

cost could be at least partly shifted towards the content consumer. A peer-to-peer network can make a business proposition for streaming content more attractive compared relying on a hosted streaming service.

Generally, a peer-to-peer network that prefers local peers to such many hops away leads to better utilization of the total bandwidth capacity of the Internet and relieves the backbone. Comparing transports between random hosts in real-life networks to streams originating from servers in well-controlled environments, one, however, quickly realizes that a number of obstacles need to be overcome. In the next section, we give an overview of these obstacles. As a possible solution, we propose a novel architecture in section 3; section 4 concludes.

2 Peer-to-Peer Streaming Media Problems

2.1 Node transience

Typically, the computers of which a peer-to-peer network is comprised are in different administrative domains; thus, they might enter or leave the network at any given time. The worst case of a user shutting down his computer while it is serving data occurs quite frequently and provides the person consuming the stream with a very bad user experience if not addressed appropriately. According to [6], roughly half of all sessions (i.e. from the time when a node registers in a network until the time it leaves it) in popular peer-to-peer networks are of 60 minutes and less.

2.2 Uplink restrictions

Available uplink bandwidth of the computers serving content is of great importance because the stream will be interrupted if a certain data rate cannot be achieved. While an Internet user in a corporate or academic environment may not experience significant service degradation when the uplink is used by a peer-to-peer application, many users connect to the Internet through a consumer broadband connection such as cable or DSL. These connections are problematic because they are typically characterized by an assymetry in their respective uplink and downlink bandwidth capacity.

As our goal is to provide the user with the best possible experience, we want to use as much of the available downlink capacity as possible for better quality of the media stream. One could consider streaming only from computers with sufficient uplink capacity, but this would severely limit the number of possible streaming sources and be considered unfair as well: a large number of users would leech bandwidth from a relatively small set of high-bandwidth users. In addition to that, traffic shaping is on the rise with ISPs, limiting individual connections to a certain rate that is less than the maximum available link bandwidth.

2.3 Firewalls and NAT

Another problem that is not particular to streaming applications, but to peer-to-peer networks in general are firewalls and network address translation (NAT) which block some connections[2].

A trivial algorithm that keeps trying until a connection can be established and initiates push requests, such as implemented in some peer-to-peer file sharing applications [7], will not lead to a good user experience because of the large delays associated with this method (moreover, it make synchronisation exceedingly difficult – we will show later why this is important). As a possible solution to address these issues we propose a topologically aware multi-source streaming network for content delivery.

3 Proposed Architecture

Our goal was to design an efficient architecture for feeding nodes with a particular multimedia stream from a set of nodes in a peer-to-peer network, and to make that stream reliable enough to achieve satisfactory QoS in real-life public networks. In our scenario, good QoS is characterized by fast connection setup and a minimization of stream interruptions. Our architecture encompasses the following set of features:

3.1 Local caching

In order to provide a large number of client nodes with content streams, a large enough number of nodes needs to have the data available locally in order to provide a transport stream for another node. This is achieved by retaining the data of received streams in a local cache (comparable to caches in web browsers) for a given minimum time-span. As a result, the total capacity of the peer-to-peer network to stream a particular file grows with the number of nodes that host that file [8].

3.2 Multiple sources

The first serious obstacle to overcome is that of limited uplink bandwidth of peers. The obvious approach to this is to combine multiple transport streams

[2] Corporate firewalls are typically configured to allow inbound connections to a certain set of defined hosts only. In most cases, these hosts will be mail servers or web servers and not machines which are part of a peer-to-peer network. The same applies for personal firewalls which are installed at the end user's computer or even a part of its operating system. Most importantly for streaming applications which typically rely on UDP based transport, the vast majority of firewalls are configured to block incoming UDP traffic. Certain types of NAT make machines that are behind the gateway completely inaccessible from the public routing infrastructure for inbound connections.

from several peers to one content stream to be received at the client. A trivial algorithm would be to interleave bytes or fixed-size segments from the sending peers, i.e. sending byte $X \ mod \ N + I$ where N is the number of sending peers and X is the byte index in the stream from the sender identified by the index $0 \leqq I < N$. This by itself would, however, just make the resulting content stream very unreliable, since the failure of any sending node would lead to a breakdown of the content stream. Due to the properties of nodes described earlier, such as transience, one would be rather lucky to experience an uninterrupted streaming session for an extended time.

3.3 Forward Error Correction

We propose to introduce redundancy into the data stream to make the resulting stream more reliably across multiple sending nodes. While forward error correction (FEC) is more frequently found at the data link layer, we believe that closely interleaving our data streams may enable us to advantageously use FEC algorithms based on block codes (e.g. Reed-Solomon) at the application layer. The media stream is first encoded at each sending node with the chosen FEC code and the resulting stream is then transmitted interleaved as described above. This should result in a reliable media stream that is not interrupted even if one or several (depending on the choice of FEC code) of the sources fail. For instance, a $(5, 1)$ Reed-Solomon code would allow one source from six to fail at a given time without breaking the media stream, at a cost of 20 per cent excess bandwidth.

The optimum code to choose for our scenario depends on a variety of factors, such as the number of sending peers, the nature of network connections, the available bandwidth at each peer, node transience and the amount of redundancy required to offset resulting transport failures (this, in turn, depends on the protocol used for transport and its behaviour with regard to session handling, congestion behavior, jitter, and so on). Another critical factor is the amount of extra data, as there exists a trade-off between FEC effectiveness and wasted bandwidth. We believe that we will be able to fine tune this parameter on-the-fly by feeding back the perceived quality of the codec the sending peers.

3.4 Recovery and lookup service

The content stream would still break, however, if more sending nodes failed than accounted for in the chosen FEC code. This can be prevented by substituting the data stream of a failed source node with another source during the lifetime of the streaming session. A low-latency lookup mechanism is required for this to rapidly identify other nodes that have the required data available. Candidates for such a service include refering to a central index of nodes and locally kept files at each node, a set of "SuperNodes" that keep this information for neighbor nodes (as implemented in the FastTrack peer-to-peer stack [3]), broadcasting a query and purely non-hierarchical lookup mechanisms such as in Gnutella [12].

For the purpose of good QoS with respect to the time required to set up a stream, a centrally kept index yields satisfactory response times for the setup of a streaming session or the quick substitution of a failed source while the session is in progress. Since such an index needs to accurately reflect the current state of the peer-to-peer network, a central instance should periodically check whether nodes have expired with a heartbeat mechanism, and nodes should report changes to the data in their local cache when they occur.

On the other hand, introducing a central index (yet another single point of failure) is clearly an architectural disadvantage. We are presently considering whether a hierarchical non-central network such as implemented in the FastTrack stack would be feasible for this purpose.

3.5 Locality and breadth first search in the network graph

The fewer hops there are between two nodes, the fewer things there are that can fail on the path between them. It has been documented that locality provides for more reliable transport in peer-to-peer networks and in fact many such network stacks implement some sort of locality (mostly measured by packet round trip times, i.e. ping, as in Gnutella, and some by hop counting, such as reportedly in Kontiki). This matters not only for reliability, but also for link bandwidth – the slowest link on the path determines the maximum bandwidth and the total network bandwidth capacity that can be utilized, since fewer connections will share the same paths such as backbone connections. It is particularly important to avoid bandwidth congestion at backbone connections when flash crowds demand popular content.

On the other hand, if the peers selected for a session share a common network link then the failure or congestion of that link will lead to the failure of several transport streams and probably break the content stream. This could probably be avoided by mapping the peer network and building a network graph [13] and using a breadth-first algorithm originating from the receiving node to search for suitable peers, or to avoid peers from the same subnet or autonomous system.

3.6 Copyright management

The commercial application of the proposed architecture for the distribution of copyrighted works can requires effective access control to content distributed through it. In the traditional server-based streaming scenario, this is trivial, because each request for a streaming session can be easily authorized or denied by a central authority and content is not stored at the client side like in the

[3] Unfortunately, little public information exists about the FastTrack stack due to its proprietary nature – even though it is the basis for the most widely used file sharing applications at the time of this writing [9] [10]; a related concept is described as ”proxylets” in [11].

local cache. The proposed architecture, however, requires that a given file is stored in other nodes' local caches and that all copies of a particular multimedia asset are identical. Therefore, where this is a concern, digital rights management technology needs to be embedded in the file format and playback client since there would otherwise be nothing preventing a user from extracting the file from the local cache and, for instance, to watch a pay-per-view movie again without payment or to distribute that movie to other users.

3.7 Protection against content spoofing

Issues of content and/or node spoofing are of particular concern in peer-to-peer networks as they rely on the collaboration of nodes to work. For instance, it would clearly not be desirous that a malicious user could feeding a video stream with pornographic content to kids expecting to see the latest Disney movie trailer. Similarly, a group of attackers could perform a logical denial-of-service attack on the content delivery network by responding with bad data to each request. This is more of a problem in streaming applications than download applications because in the latter context is feasible to verify the downloaded file against a cryptographically secure hash (such as SHA-1) from a trusted source. This method can only be applied once large file segments have been completely downloaded, requiring a very small amount of known good data to verify the authenticity of a large file. In a streaming application, this is not possible because one would have to verify each video frame or segments of audio of a certain length against a known good hash value that would need to be transferred in advance from a trusted source.

If multiple transport stream sources are introduced into the architecture, however, the attacker would need to control all source nodes to successfully introduce bad content into the network, and more than one node in a particular transmission for an effective denial of service attack for that session. Therefore, a multi-source streaming architecture is likely more secure compared to a peer-to-peer streaming network with a single source peer only.

4 Conclusions and Future Work

There are many issues left open by this proposed architecture. A very important one is the optimum transport protocol between the nodes. Candidates we are considering are RTP/RTSP, TCP-RTM ([14]), TCP, and HTTP – the last two primarily to enable streams to pass through common firewall configurations. We believe that a fallback mechanism should be implemented to discover the best possible protocol allowed by a particular configuration. More research is needed on selecting a FEC algorithm and the best suited code for combining multiple streams. A protocol needs to be designed for negotiate FEC encoding and to synchronize multiple transport streams. Lastly, some empirical research is needed to assess the real-life performance and benefits of this architecture that we hope for. We also believe it would be interesting to apply the multiple-source concept to application-level multicasting for improved reliability.

Related projects include the SpreadIt architecture developed in [15], which distributes bandwidth requirements across the network for application-level multicast. The Content-Addressable Web concept seeks to create a foundation for ad-hoc content delivery with certain extensions to HTTP, as described in [16], and could serve as a mechanism to identify peers with specific content. A different, self-organizing content-addressable network is described in [17]. General approaches to efficient content distribution are discussed in [18] and [19]. Finally, a variety of commercial applications has been developed to distribute multimedia content over peer-to-peer networks [20] [21], underlining the increasing demand for such services.

References

1. Sally Floyd and Kevin Fall, *Promoting the Use of End-to-End Congestion Control in the Internet*, IEEE/ACM Transactions on Networking, August 1999.
2. Tyron Stading, Petros Maniatis, Mary Baker, *Peer-to-Peer Caching Schemes to Address Flash Crowds*, 1st International Peer To Peer Systems Workshop (IPTPS) 2002.
3. Mojo Nation, http://www.mojonation.net/, June 2001.
4. *Adcritic.com: A Victim of Its Own Success?*, NY Times, December 20, 2001.
5. *CNN's video fees add to the Net's growing price tag*, USA Today, March 18, 2002.
6. Stefan Saroui, P. Krishna Gummadi, Steven D. Gribble, *A Measurement Study of Peer-to-Peer File Sharing Systems*, Technical Report UW-CSE-01-06-02, University of Washington, July 2001.
7. P. Backx, B. Duysburgh, T. Lambrecht, L. Peters, T. Wauters, P. Demeester, B. Dhoedt, *Enhanced applications through active networking*, 2nd FTW PHD Symposium, Interactive poster session, paper 99 (Proceedings available on CD-Rom), 12 December 2001, Gent, Belgium 2001.
8. *Kontiki Technology: The Mojo*, available at http://www.kontiki.com/technology/, 2002.
9. *Napster Shutdown Feeds Dutch Peer-To-Peer Startup*, InformationWeek, July 16, 2001.
10. Joe St. Sauver, *Percentage of Total Internet2 Traffic Consisting of Kazaa/ Morpheus/ FastTrack*, available at http://darkwing.uoregon.edu/∼joe/kazaa.html, 2002.
11. Atanu Ghosh, Michael Fry, Jon Crowcroft, *An Architecture for Application Layer Routing*, Yasuda, H. (Ed), Active Networks, LNCS 1942, Springer, pp 71-86. ISBN 3-540-41179-8 Springer-Verlag.
12. Evangelos P. Markatos, *Tracing a large-scale Peer to Peer System: an hour in the life of Gnutella*, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.
13. Bradley Huffaker, Daniel Plummer, David Moore, and k claffy, *Topology Discovery by Active Probing*, available at http://www.caida.org/outreach/papers/2002/SkitterOverview/
14. Sam Liang and David Cheriton: *TCP-SMO: Extending TCP for Medium Scale Multicast Applications*, IEEE Infocom 2002, June 23-27 2002, New York.

15. Hrishikesh Deshpande, Mayank Bawa and Hector Garcia-Molina, *Streaming Live Media over a Peer-to-Peer Network*, Stanford Technical Report 2001-30.
16. Justin Chapweske, *HTTP Extensions for a Content-Addressable Web*, available at http://onionnetworks.com/caw/, 2001.
17. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Schenker, *A Scalable Content-Addressable Network*, ACM SIGCOMM 2001, San Diego, CA, August 2001.
18. Sylvia Ratnasamy, Mark Handley, Richard Karp, Scott Shenker, *Topologically-Aware Overlay Construction and Server Selection*, IEEE Infocom 2002, New York, June 23-27 2002.
19. Y. Chawathe, S. McCanne, and E. Brewer, *An Architecture for Internet Content Distribution as an Infrastructure Service*, Technical report, available at http://www.research.att.com/∼yatin/publications/, 2000.
20. *Peer-to-Peer Streaming Finds Friends*, Streaming Media, August 28, 2001.
21. *AllCast Increases Profits for Streaming Media* (Press Release), August 16, 2001.