Scalable Peer-to-Peer Streaming for Live Entertainment Content

Eleni Mykoniati, Raul Landa, Spiros Spirou, Richard G. Clegg, Lawrence Latif, David Griffin, and Miguel Rio, University College London

ABSTRACT

We present a system for streaming live entertainment content over the Internet originating from a single source to a scalable number of consumers without resorting to centralized or provider-provisioned resources. The system creates a peer-to-peer overlay network, which attempts to optimize use of existing capacity to ensure quality of service, delivering low startup delay and lag in playout of the live content. There are three main aspects of our solution: first, a swarming mechanism that constructs an overlay topology for minimizing propagation delays from the source to end consumers; second, a distributed overlay anycast system that uses a location-based search algorithm for peers to quickly find the closest peers in a given stream; and finally, a novel incentive mechanism that encourages peers to donate capacity even when the user is not actively consuming content.

INTRODUCTION

Live content delivery over the Internet is highly desirable, but challenging. Removing the need to deploy specialized resources would allow anyone to broadcast and reach consumers anywhere in the world. This facility could be used by many applications — the most obvious being IPTV for the distribution of live audio-visual content, but other applications such as real-time multi-user 3D environments and games could also be envisioned.

While multicast and quality of service (QoS) mechanisms at the network layer seem ideal for delivering live entertainment content, deployment is still limited. Lack of multicast can be overcome, although not efficiently, by using unicast connections to consumers. Lack of network layer QoS, however, inevitably impacts quality of experience (QoE) observed as interruptions to the playback of many currently deployed client/server applications. Borrowing from traditional content delivery networks (CDNs) for Web content, IPTV service providers currently fence off a small portion of the Internet and deploy native multicast, traffic differentiation, abundant capacity, and dedicated infrastructure. Although this *walled garden* gives acceptable results, even for 8 Mb/s high definition (HD) H.264 TV channels, it is neither scalable nor cheap.

Application-layer multicast-like solutions typically rely on organized distribution trees [1], so they are sensitive to *churn* (frequent peer arrivals and departures). On the other hand, file distribution systems like BitTorrent [2] achieve high resilience to churn with *swarming*: splitting the file into small data units and ensuring that these pieces are distributed among the set of peers participating in the download. In this way peers obtain parts of the file from many peers in parallel, increasing resilience. However, content delivered in this way can only be consumed once the full download has been completed.

Variations of BitTorrent for on-the-fly consumption of swarmed content such as PPLive [3] suffer from relatively long startup delay and playout lag. In addition, they often require provider-provisioned resources (super peers) to assist content distribution by providing supplementary upload capacity beyond the capacity provided by the consuming peers.

Our solution is a peer-to-peer live streaming system where all resources are provided by the peers themselves. All peers participate in the distribution and consumption of one or more streams produced at a single source — the *peercaster*. Upon joining a stream, peers attach themselves to other peers according to distributed topology management and capacity allocation algorithms that aim at minimizing startup delay and playout lag across the entire swarm.

The article is organized as follows. First we discuss the problems to be addressed by a swarming system delivering real-time content. The following sections then describe the main aspects of the system: the construction of an overlay network to deliver swarmed media streams with QoS considerations; the discovery of peers participating in a stream in a scalable way; the participation of non-consuming peers to increase the capacity of the system; and an incentive mechanism that enables claiming past contributions from peers with no prior interaction. Following the description of the main components of our solution, we describe how they fit together to form an overall system for streaming live media content over a fully distributed peer-to-peer network. The final section presents our conclusions.

0163-6804/08/$25.00 © 2008 IEEE

REAL-TIME SWARMING

Peer-to-peer live streaming systems overcome the lack of native multicast and QoS by implementing such functionality at the application layer. The resultant *overlay* receives the live stream from an *encoding element* (e.g., a capture device) at the peercaster, distributes it through the peers, and delivers it to a *decoding element* (e.g., a player) at the *consumer*. The system performance generally differs across consumers and is correlated to QoE through three metrics:

- *Startup delay*: The time between a consumer's request to view a particular stream and the stream beginning to play. Low startup delay allows quick switching between channels.
- *Playout lag*: The delay between a stream data unit being sent by the peercaster and the same data unit being viewed by the consumer. Low playout lag means that stream viewing is more "live."
- *Playout continuity*: The percentage of stream data units successfully played at the correct time by the consumer. High playout continuity enables viewing with minimal distortion.

Early peer-to-peer streaming systems mimicked native multicast and organized peers in a *tree* with the peercaster at the root. After an initial consumer request, the entire stream would be *pushed* to the consumer from a peer with available upload bandwidth and probably in close network proximity [1]. This scheme gives good startup delay and playout continuity if the tree is stable under churn, and good playout lag if the tree is also short (low number of peers between peercaster and consumer). The highly dynamic Internet environment has so far kept *tree/push* systems within research settings.

Recent deployments of peer-to-peer live streaming systems [3] are based on the fundamentally different and inherently resilient *mesh/pull (swarming)* scheme: a stream is segmented at the peercaster into data units called *chunks*, and each consumer independently optimizes its playout by selecting which chunks to *pull* from which peers. The resultant interconnection graph, as shown in Fig. 1, is not a regular grid or fully connected, but it is called a mesh nonetheless. Mesh/pull streaming systems have built on the success of the BitTorrent file distribution mechanism, and owe their resilience to local optimization of delivery and free selection of peers and chunks. Similar to BitTorrent, the set of peers exchanging chunks constitute a *swarm*. Unlike BitTorrent, chunks must leave the overlay and reach the decoder at the consumer *in order* and *in real time*. Furthermore, unlike video on demand (VoD), all chunks are not available at the time a peer selects the stream; future chunks can only be distributed by the peercaster after they have been generated.

A mesh/pull peer requests and receives chunks from different peers in the swarm. To absorb the variance in chunk reception over different sources and cope with peer failures in times of high churn, a consumer maintains a *playout buffer*. The playout buffer contains chunks yet to be played out and slots for chunks yet to be retrieved. It is marked by the playout point, which indicates the next chunk to be delivered to the decoder. Playout begins once the number of received consecutive chunks is considered sufficient to support playout continuity at the stream rate, marking the startup delay and initial playout lag.

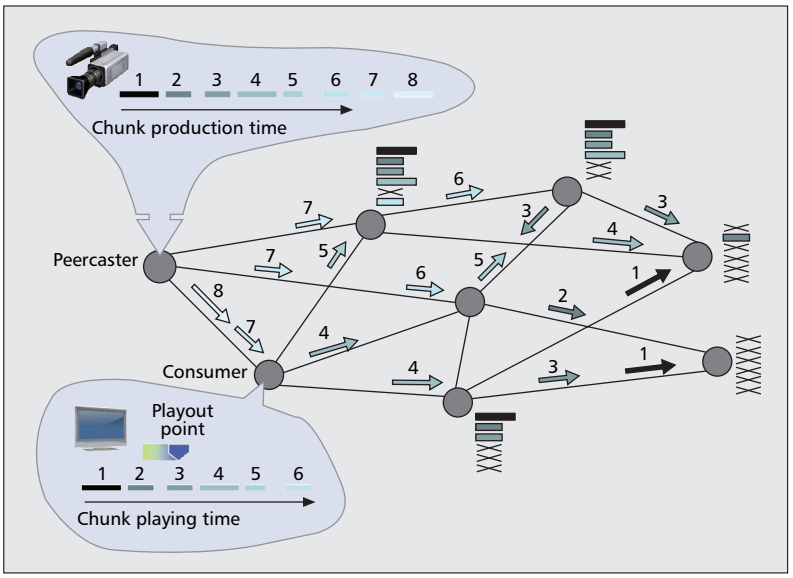


Figure 1. *Real-time swarming.*

A large playout buffer increases playout continuity because it absorbs more jitter, but it also incurs higher startup delay and playout lag. The variance experienced in chunk reception is typically smaller for less live chunks, as these are already distributed across a large set of peers. More live chunks, on the other hand, are harder to find, so retrieving them in time to preserve continuity requires a very aggressive strategy. The particular choice in the trade-off between playout lag and playout continuity depends on the capabilities and position of each peer in the swarm, and must follow the preferences of the user; it is therefore subject to local optimization by the peer.

The *best* performance of currently deployed swarming systems can be summarized as having 20 s startup delay and 1 min playout lag with acceptable playout continuity for streams of 350 kb/s [4]. This is far from even the *average* performance of current IPTV walled gardens that exhibit 2 s of startup delay and 2 s playout lag with good playout continuity for streams of 3.5 Mb/s.

The following section discusses principles for selecting peers from which to retrieve chunks, with the objective to improve both playout lag and continuity for all peers in the system, irrespective of their position in the trade-off between playout lag and playout continuity.

A QOS-BASED OVERLAY

The data connections for retrieving chunks, determined by the local strategies of the individual peers, form an overlay network. The delay and throughput (QoS) over each connection and along the end-to-end path have direct impact on the QoE experienced by each consumer. The

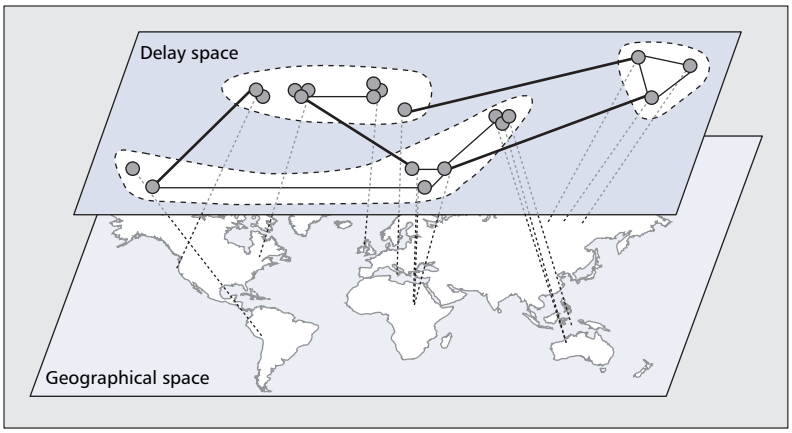


Figure 2. *Mapping peers to synthetic delay coordinates.*

total throughput achieved over all the incoming connections of a consumer must be enough to sustain the stream rate to ensure playout continuity. Also, minimizing the playout lag is achieved by minimizing the delay over the end-to-end path from the peercaster to the consumer. We define this *path delay* as the accumulated link propagation and processing/queuing delays at each sending peer.

A poor overlay topology is one where traffic traverses long overlay links disregarding direction or locality. In addition to the negative impact on playout lag for the peers, this also increases traffic in the underlying network. If the overlay is designed more intelligently by interconnecting local peers whenever possible, this has the side-effect of benefiting the Internet service providers (ISPs), who will experience less traffic over expensive interdomain links.

Peers need an estimate of network delay to other peers to determine which ones are in proximity. It is impractical to establish a connection, probe, and measure the delay to every peer before selecting only the nearest few. To address this issue, recently developed techniques can obtain *delay space coordinates* based on only a few prior measurements [5]. Network delays are mapped to a synthetic coordinate space using metric space embedding (Fig. 2); delay between any two peers can be approximated with their distance in delay space.

A star (client/server) topology where all peers directly connect to the peercaster is ideal for minimizing delay. However, this is unrealistic as it implies that the peercaster has sufficient upload capacity to serve all peers. The best approximation of ideal direct connections to the peercaster is achieved when peers establish connections with their closest peers in the direction of the peercaster. Always favoring short connections, though, results in paths with a large hop count, and thus potentially long delays due to accumulated processing and queuing times at each hop. Our simulation results [6] indicate that a compromise of establishing a few longer connections to serve as "jumps" toward the peercaster is beneficial in terms of reducing hop count without requiring increased capacity at the peercaster.

Peers are not uniformly distributed across the delay space. Rather, they tend to cluster following population density modified by the Internet topology and associated delays. Connections within the same domain have small delays, whereas one satellite interdomain link is enough for the delay to soar. Peers that only select fast local connections may fail to reach other peers that are close enough to the peercaster to actually get the stream. Therefore, a cluster of peers needs to have enough long-distance (slow) connections to reach other remote peers with the required upload capacity and access to the stream data.

Once a peer has determined the set of feasible peers it prefers on topological and delay grounds, it will usually request and receive data from only a subset. Decisions on from which peers to download a particular chunk depend on the potential senders' playout lag and chunk availability, and other factors such as the actual network delay experienced over the links, their available upload bandwidth, stability, trust, and past performance.

The following section elaborates on the distributed discovery of peers to establish the required connections in the direction of the peercaster.

Scalable Peer Discovery

A critical aspect of our system is the search for other peers that are available to send chunks of the stream. When first connecting to a stream, a peer wishes to quickly determine which other peers can provide chunks and are also close in terms of delay. To achieve this we introduce the concept of a *local tracker* (LT). Local trackers are ordinary peers that have the additional function of listing peers carrying a stream in their *local area*. The scope of local areas is dynamic: they split or merge as they become over-/underloaded as peers join/leave the stream. Figure 3 shows an example of three streams, their local areas, and the underlying distribution of nodes on all streams.

As already mentioned, low startup delay is an important goal for the system and involves peers being able to quickly obtain a list of nearby peers carrying the stream in which they are interested. To achieve this task we developed a distributed overlay anycast table (DOAT) inspired by the Chord DHT system [7].

While an LT maintains the list of peers carrying a particular stream in the local area, the DOAT maintains the list of all the LTs for all the streams across the entire network. Only a small set of peers that are stable across all the streams become DOAT nodes. When queried by a peer about a particular stream, the DOAT has to quickly search for the LT which is near that peer in terms of delay. The discovered LT then replies to the querying peer with the list of peers in its local area (the related interactions are discussed later).

To facilitate the search task within the synthetic coordinate domain, network coordinates are transformed to a single dimension. This is achieved using a space-filling curve which has the property that if two locations are "close" on the curve, they are also close in the original space (but not vice versa). One particular curve

that has been shown to have this property is the *H curve* [8]. Figure 4 shows a space-filling curve generated with three iterations and how nodes map to it. In our system this space-filling curve replaces the ring structure of a typical Chord DHT; thus, the distance in the DOAT ring structure corresponds to the distance in the network delay space (with the potential error introduced by the space-filling curve).

The routing table of a DOAT node is, similar to Chord, populated with neighbor DOAT nodes at exponentially increasing distances in either direction of the DOAT ring. A DOAT routing entry contains the next-hop DOAT neighbor node and the list of identifiers of the streams for which LT data can be reached through this DOAT neighbor node. To accelerate the process of matching a stream identifier to a list and reduce the overhead of routing table updates, the list of stream identifiers is aggregated into a *Bloom filter* [9]. In this case the Bloom filter is the result of setting to logical 1 all the bits corresponding to each of the k hash values of all the identifiers in the list, and provides an efficient data structure to store and verify set membership.

Whenever a new LT for a particular stream is added, it registers with the closest DOAT node in delay space, and the stream identifier is added to the Bloom filter of the *local* routing entry of that DOAT node. Furthermore, if the DOAT node had no previous route for the stream identifier of the newly added LT, a routing update is sent to all its neighbors, signaling the existence of the new route. Upon receiving a routing update from neighbor node *A*, DOAT node *B* updates the routing entry associated with *A* and further forwards the update to its neighbors that are further away, in terms of their distance along the space-filling curve, than node *A* (closer nodes will receive the update from nodes nearer to them). In order to reduce the frequency of messages, a minimum interval is introduced over which routing updates are aggregated and sent over a single routing update message.

To obtain the IP address of the closest LT carrying a given stream, peers issue a query with the stream identifier to their closest DOAT node. The DOAT node then searches its routing table in increasing order of distance, and the query is forwarded to the next-hop DOAT node of the first routing entry whose Bloom filter matches the stream identifier. This is done recursively at each DOAT node, and the query propagates in logarithmically decreasing distances until reaching the DOAT node with an LT for the requested stream identifier in its local routing entry. The LT discovered by this process is the closest LT to the querying peer in the requested stream.

We have evaluated DOAT in a network of 1000 DOAT nodes, distributed uniformly across a coordinates space with 104 ms average delay between any two points. For a 100-LT stream, the average query time is 28 ms, while the equivalent time to a central server would be the average round-trip time (208 ms). Unlike a central server system, DOAT scales with the number of peers.

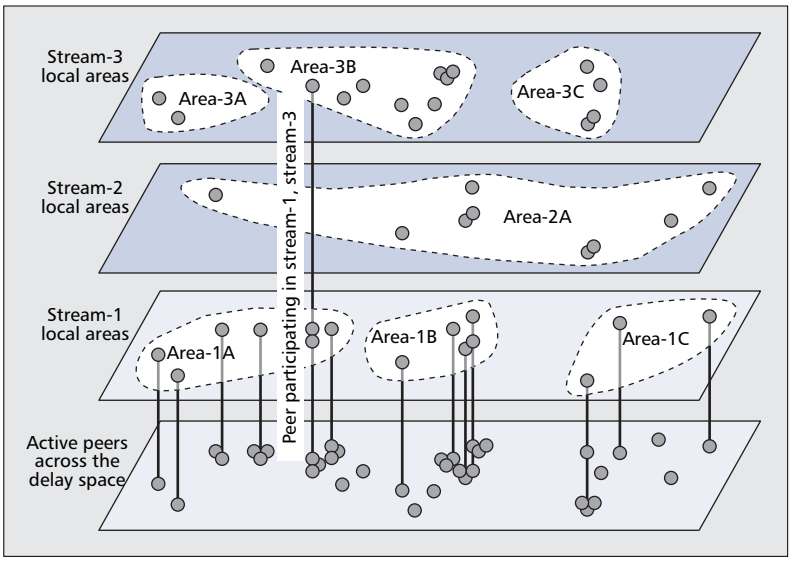
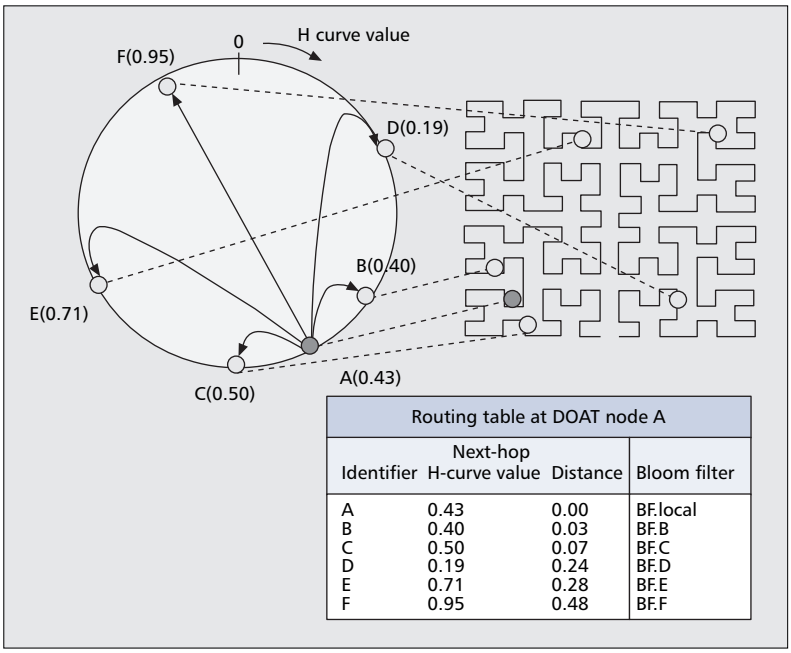


Figure 3. *Peer clustering in local areas across multiple streams.*



Routing table at DOAT node A

Identifier	Next-hop H-curve value	Distance	Bloom filter
A	0.43	0.00	BF.local
B	0.40	0.03	BF.B
C	0.50	0.07	BF.C
D	0.19	0.24	BF.D
E	0.71	0.28	BF.E
F	0.95	0.48	BF.F

Figure 4. *Distributed overlay anycast table.*

Distributed Capacity Provisioning

Traditionally, QoS for multimedia services is provided through capacity overprovisioning and service differentiation. These are impossible for large-scale real-time video distribution overlays with no control over their underlying infrastructure. Instead, the idea is to exploit the large aggregate upload bandwidth of the system to serve the consuming peers. This is possible assuming that some peers will participate in the system without consuming stream content. We call these non-consuming peers (NCPs). Through our incentives mechanism (see the next section), NCPs contribute to the distribution of chunks in order to "redeem" these contributions later by "purchasing" chunks for their own consumption.

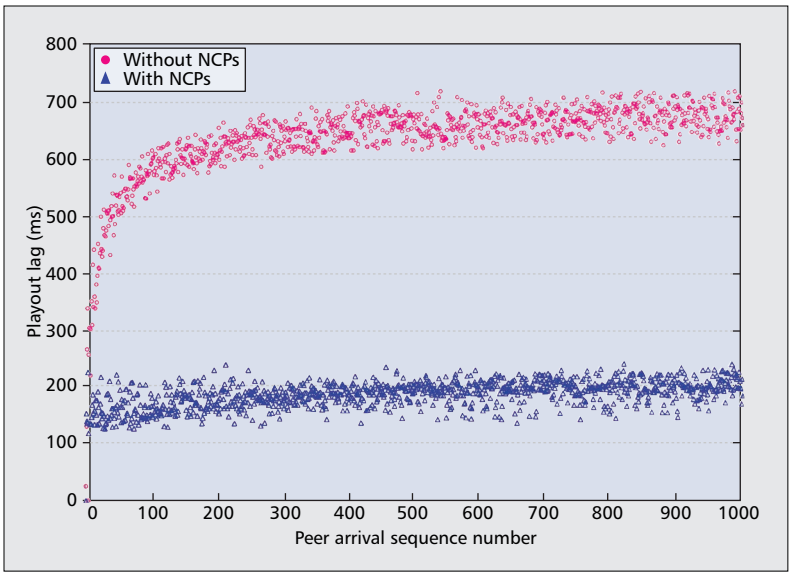


Figure 5. *Performance enhancement with non-consuming peers.*

When an NCP downloads a chunk, it uses the upload bandwidth of another peer. To offset this loss of system resources and further optimize distribution, an NCP conducts bandwidth multiplication by downloading only a fraction of a stream's chunks and uploading them as many times as possible. Thus, NCPs increase the QoS for consuming peers instead of just distributing media traffic between themselves.

Another advantage of NCPs is reduction of the playout lag when they are deployed appropriately. This can be accomplished by encouraging NCPs, at stream bootstrap, to join in order of their delay from the peercaster. The result is an interconnected web of bandwidth multipliers with low playout lag. Consuming peers inherit this low playout lag when they connect to a stream containing such organized NCPs.

This effect can be seen in Fig. 5, depicting the playout lag for a simulation of 100 NCPs and 1000 peers, distributed uniformly across a delay space with 76 ms average delay between any peer and the peercaster. We model a 1.5 Mb/s rate stream split into five substreams, each of which has the same bandwidth. All peers have sufficient download capacity and an upload capacity of 2.1 Mb/s. NCPs join at stream bootstrap and subscribe to only one substream.

INCENTIVES

As has been shown repeatedly [10], peer-to-peer systems where there is no contribution accounting and associated incentive mechanism can suffer from pervasive freeloading and widespread quality degradation. Additionally, as detailed in the previous section, a QoS-based overlay network requires peers to contribute resources to the system even if they are not interested in consuming resources at the same time. This means that peers must be able to make contributions toward some set of peers, and then redeem those contributions at a later time from a different set of peers.

This problem is usually solved using reputation systems [11], where peers implement extended "word-of-mouth" recommendation networks. However, most reputation systems are vulnerable to identity-based attacks where a single peer commands legions of disposable identities, or peers lie about the contributions of other peers in an attempt to deny them access to network resources. We propose an incentives mechanism where peers only keep track of the contributions they have given or received to/from specific neighbor peers, but are able to use with all peers. Additionally, our protocol is designed to ensure that peers are unable to profit from multiple identities and do not need to rely on possibly false third-party information about the contributions of other peers.

In order to address these identity management issues peers use signed pseudonyms that their neighbors can verify with public keys exchanged when the peers initially come into contact. From then on, all communication between peers is digitally signed. Thus, contributions are always bound to an identity, and peers gain nothing from having multiple identities: their contributions will simply be split among them. Additionally, we propose the translation of contributions to an abstract numerical trust value, which can be directly manipulated by the peers and reconverted into contributions when necessary. The trust peer A has with peer B corresponds to the contribution peer A has received from peer B in the past without offering anything in return, and ought therefore to return to peer B in the future. Note that there is no explicit interaction for payment. Instead, each peer maintains trust from/to other peers locally.

To enable peers to contribute to a set of peers but receive contributions from another, we propose *trust shifting*. Trust shifting enables a peer to receive a contribution from a peer with which it has no previously established trust, by shifting trust along a path of trust relationships among other peers. Thus, any peer can provide contributions to a given peer and then request to have this trust shifted to a different peer from which it can now request any service.

To accomplish this, each peer constructs a local view of the *trust network*. Its immediate neighbors are found by direct experience, but neighbors further away are discovered using a truncated self-avoiding random walk algorithm that discovers high-trust paths preferentially. In essence, each peer periodically advertises its local trust accounts. As these advertisements necessarily traverse the peers who actually performed the contributions represented in their account values, it is trivial to ensure that only truthful messages are propagated. Messages are propagated preferentially through links with high trust in a probabilistic fashion. The probability of messages being forwarded to a given neighbor peer is proportional to the local trust account of this neighbor. Thus, an announcement has greater probability of traversing paths where trust links have consistently high trust values.

In order to determine the maximum amount of trust that can be shifted to any particular destination along the discovered trust connections, each peer runs an instance of a MaxFlow-MinCut algorithm [12]. The shift itself is implement-

ed through trust shift request messages that are source-routed and execute the trust exchanges on a hop-by-hop basis. Once trust has been shifted, it is essentially indistinguishable from trust acquired by direct contribution, allowing peers to flexibly decide where to "spend" their accumulated trust.

Since peer-to-peer streaming is a delay-sensitive system built over the best effort Internet, peers are unable to determine whether long delays are due to the queuing at their serving peer or the congestion across the overlay link between them. We address this by enabling peers to draft quality-dependent contracts [13] and conduct payments through trust shifting.

Putting It All Together

As discussed in the previous sections, the system consists of multiple independent swarms carrying different streams of real-time content. Peers participate in one or more streams when they wish to consume or just distribute their content. To participate in a stream, a peer must discover and connect to other peers in the swarm carrying it and exchange chunks of content. Knowledge about other peers is obtained from a distributed set of per-stream LTs, and knowledge about the LT per stream is maintained by the DOAT system. The following paragraphs illustrate the interactions between our system components and the actions necessary for a peer to receive swarmed content (Fig. 6).

When a peer wishes to join the system and subsequently participate in particular streams, its first task is to discover its closest DOAT node. The peer must first position itself in delay space by performing a series of delay measurements to a set of locations, as in Vivaldi [5], and use these to calculate its synthetic coordinates. Armed with these coordinates, the peer queries any known node on the DOAT overlay for the closest DOAT node in the synthetic coordinate space (A1, Fig. 6). The DOAT node receiving the query forwards it to the node closest in delay space to the querying peer (A2, Fig. 6), and the identifier of the nearest DOAT node is returned (A3, Fig. 6). This is the point of contact for all subsequent queries made by that peer, and because it is the closest, it is the best in terms of minimizing stream startup delay for that peer.

Before joining a particular stream, a peer must first find its LT. It is assumed that peers discover the identifiers of the streams they wish to view through an out-of-band process (e.g., Webpage hyperlinks, electronic program guide, or gossip through a social network). The peer uses the stream identifier to query the DOAT node identified in the previous steps for the closest LT for that stream (B1, Fig. 6). The DOAT system then routes the query to the closest LT, as described earlier. The query is first routed to the DOAT node closest to the LT closest to the peer (B2, Fig. 6), who subsequently forwards the request to the LT (B3). The LT then registers the peer in its database, together with its coordinates and other attributes such as upload capacity, and returns the list of registered peers back to the peer who originated the query (B4, Fig. 6). At this point the peer has a list of all the peers participating in the desired stream in the local area together with their attributes, such as coordinates in delay space and capacity.

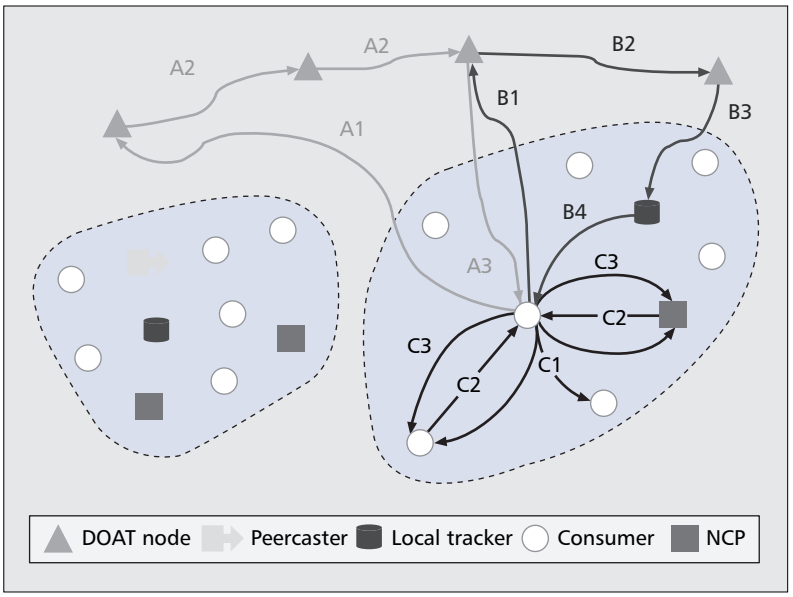


Figure 6. *Peer interactions overview.*

Given the list of peers participating in the stream and their attributes, the peer can determine to which peers it should connect according to the principles of real-time swarming over a QoS-based overlay, outlined previously. Peers issue requests for chunks to other peers (C1, Fig. 6). The chunks are transmitted to the requesting peer (C2, Fig. 6), and payments are made (C3, Fig. 6). Payments are shown in the figure as being passed in the direction of receiver to sender, although this is only a logical transfer as trust is the basis of our incentive mechanism, and the establishment of trust following any transaction happens in the opposite direction: the receiver has more trust in the sender after a chunk has been received.

Figure 6 also shows an example of an NCP supplying the peer with chunks. The NCP receives payment (actually the NCP builds trust in the peer receiving the chunk), which is stored and can be used in other streams at other times, thanks to the trust shifting mechanism described in the previous section, to pay for chunks in those other streams and therefore improve experienced QoS. Although the peer acting in the NCP role does not benefit directly by consuming the content of the stream, it is incentivized to participate in the local area shown in Fig. 6, increasing the capacity of the overlay for that stream and reducing overall playout lag for the peers. This is an example of the incentive mechanism working in practice.

Conclusions

This article has discussed the problems of delivering real-time content over the Internet and has outlined a novel approach for swarming such content over peer-to-peer networks in a scalable manner without resorting to centralized resources. Synthetic network coordinates model

We have discussed the problems of delivering real-time content over the Internet and have outlined a novel approach for swarming such content over peer-to-peer networks in a scalable manner without resorting to centralized resources.

the position of peers in delay space, and are used to assist peers in building a QoS-aware overlay network, and schedule chunk requests and transfers over the overlay so as to reduce playout lag and increase playout continuity. Scalability and resilience are enhanced by the use of distributed local trackers, which are in turn managed by a novel distributed overlay anycast table. The design of the DOAT overlay and the use of local trackers are to ensure that queries are resolved quickly, so startup delay is kept small. Non-consuming peers are encouraged to participate in streams by use of an incentive mechanism that allows contributions by a peer to be rewarded in other streams at different times. This is achieved in an entirely distributed manner without requiring a central currency repository. We have demonstrated that NCPs not only increase the available bandwidth to a swarm but also reduce playout lag.

References

[1] V. Venkataraman, K. Yoshida, and P. Francis, "Chunkyspread: Heterogeneous Unstructured Tree-Based Peer-to-Peer Multicast," *Proc. IEEE Int'l. Conf. Network Protocols*, K. Yoshida, Ed., Santa Barbara, CA, 2006, pp. 2–11.

[2] B. Cohen, "Incentives Build Robustness in BitTorrent," *Proc. Wksp. Econ. Peer-to-Peer Sys.*, Berkeley, CA, June 2003.

[3] X. Hei *et al*., "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Trans. Multimedia*, vol. 9, Dec. 2007, pp. 1672–87.

[4] A. Sentinelli *et al*., "Will IPTV Ride the Peer-To-Peer Stream?," *IEEE Commun. Mag.*, vol. 45, June 2007, pp. 86–92.

[5] F. Dabek *et al*., "Vivaldi: a Decentralized Network Coordinate System," *Proc. 2004 Conf. Apps., Tech., Architectures, Protocols Comp. Commun*., Portland, OR, Aug. 2004, pp. 15–26.

[6] R. G. Clegg *et al*., "The Performance of Locality-Aware Topologies for Peer-To-Peer Live Streaming," *Proc. UK Perf. Eng. Wksp.*, 2008.

[7] I. Stoica *et al*., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001, pp. 149–60.

[8] R. Niedermeier, K. Reinhardt, and P. Sanders, "Towards Optimal Locality in Mesh-Indexings," *Discrete Applied Mathematics*, vol. 117, Mar. 2002, pp. 211–37.

[9] B. H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *ACM Commun.*, vol. 13, no. 7, 1970, pp. 422–26.

[10] D. Hughes, G. Coulson, and J. Walkerdine, "Free Riding on Gnutella Revisited: The Bell Tolls?," *IEEE Distrib. Sys. Online*, vol. 6, no. 6, June 2005.

[11] P. Resnick *et al*., "Reputation Systems," *ACM Commun.*, vol. 43, no. 12, Dec. 2000, pp. 45–48.

[12] A. V. Goldberg and R. E. Tarjan, "A New Approach to the Maximum-Flow Problem," *J. ACM*, vol. 35, no. 4, 1988, pp. 921–40.

[13] R. Landa *et al*., "Incentives against Hidden Action in QoS Overlays," *Proc. 8th Int'l. Conf. Peer-to-Peer Comp.*, Germany, Sept. 2008.

Biographies

ELENI MYKONIATI (e.mykoniati@ee.ucl.ac.uk) received a B.Sc. in computer science from Piraeus University, Greece, in 1996 and a Ph.D. degree from the National Technical University of Athens (NTUA), Greece, in 2003. She has worked as a research associate for Telscom S.A. Switzerland, the NTUA DB and Telecom Labs, and Algonet S.A. Greece. Since 2007 she is a research fellow in the Department of Electronic and Electrical Engineering at University College London, United Kingdom. Her research interests include business-driven traffic engineering in IP networks, QoS, and peer-to-peer networking.

RAUL LANDA (raul.landa@ieee.org) is currently a Ph.D. student in the Department of Electronic and Electrical Engineering at University College London. His research interests involve applying models from economics and sociology to networking problems. He received a B.Eng. in communications engineering from ITESM, Mexico City, an M.Sc. in data communications from the University of Sheffield, United Kingdom, and a Diploma in information security from UNAM, Mexico City.

SPIROS SPIROU (spiros.spirou@ucl.ac.uk) holds a Ptychion in computer systems engineering, a postgraduate diploma in mathematics, and an M.Sc. in information processing and neural networks. He is currently studying part-time for the Ph.D. in peer-to-peer streaming at University College London. Currently, he is a project manager and software engineer at Intracom Telecom, Greece, and a research associate at NCSR Demokritos, Greece. His research interests are broadband television, peer-to-peer networking, and network management.

RICHARD G. CLEGG (richard@richardclegg.org) is a senior research fellow in the Department of Electronic and Electrical Engineering at University College London. He gained a Ph.D. in mathematics from the University of York in 2005. His research interests include long-range dependence, queuing theory, peer-to-peer networking, and network topologies.

LAWRENCE LATIF (llatif@ee.ucl.ac.uk) is currently a Ph.D. student in the Department of Electronic and Electrical Engineering at University College London. His current research area is resilient distributed searching mechanisms and overlays. He received a B.Sc. in computer science with management from Kings College, London, and an M.Sc. in systems engineering management from University College London.

DAVID GRIFFIN (dgriffin@ee.ucl.ac.uk) is a senior research fellow in the Department of Electronic and Electrical Engineering, University College London. He has a B.Sc. in electrical engineering from Loughborough University, United Kingdom, and is currently completing a part-time Ph.D. in electrical eengineering from the University of London. Before joining University College London he was a systems design engineer at GEC-Plessey Telecommunications, United Kingdom, and then a researcher at the Foundation for Research and Technology-Hellas, Institute of Computer Science, Crete, Greece.

MIGUEL RIO (m.rio@ee.ucl.ac.uk) is a lecturer in telecommunications and computer networks at the Department of Electronic and Electrical Engineering, University College London. He holds B.Eng. and M.Sc. degrees from the University of Minho, Portugal, and a Ph.D. from the University of Kent, United Kingdom. He is the principal investigator of several research projects in the area of computer networks. His current interests are peer-to-peer multimedia, QoS routing, network measurement, and Internet topologies.