Towards the Construction of Attack Resistant and Efficient Overlay Streaming Topologies

Thorsten Strufe [1] Jens Wildhagen [2] Günter Schäfer [3]

Department of Telematics and Computer Networks
Technische Universität Ilmenau
Ilmenau, Germany

Abstract

Even though overlay streaming is an inherently fault tolerant and stable system architecture, careful neighbor selection is a significant task. Inappropriate routing decisions can lead to an unstable topology with only a few very important nodes on which a large set of succeeding nodes depend. The presented algorithm selects streaming neighbors based on local information, passing knowledge to parent nodes only. Similar to SplitStream [6], it creates inner-node disjoint multicast trees. The created topologies are broad and have short paths, thus improving the resistance to node failure and intentional attacks. A malicious node can neither gain any knowledge about different regions of the topology other than its own successors nor deliberately move to a more important position in the hierarchy. The characteristics of the created topologies are revised in a static simulation study calculating the vertex connectivity and packet loss on node disconnections.

Key words: Service Availability, Application Layer Multicast, Cooperative Streaming, Resilient Overlay Networks

1 Introduction

Content dissemination generally follows one of three main delivery techniques, client-server-unicast, network multicast or application layer multicast. The client-server system architecture is relatively easy to implement and set-up and currently the prevalent solution. However, it suffers from the draw-backs of being unscalable over the number of clients and having a single point of failure: the server or cluster of servers itself. Though network layer multicast is far more scalable due to data replication in the routers of the backbone,

[1] Email: thorsten.strufe@tu-ilmenau.de
[2] Email: jens.wildhagen@tu-ilmenau.de
[3] Email: guenter.schaefer@tu-ilmenau.de

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

it suffers from low acceptance and is not largely deployed today. This fact is due to a multitude of problems including billing issues and an inherent lack of scalability over the number of multicast groups [13].

Overlay streaming, or application layer multicast (ALM) [7], makes use of the resources at the edge of the network and thus introduces a different load balancing scheme. Due to its self-organization it is inherently resistant towards failure of and attack on nodes. This stability is increased for multi-source systems in contrast to single-source systems, which construct only one single streaming tree. In contrast, every node receives different parts of the content from different nodes, and the nodes are not necessarily reliant on a single preceding node only. Peer-to-peer systems have proven to be very resistant both towards failures and organized interference. The robustness of this system architecture becomes apparent when observing file sharing systems and the efforts to disable them.

In contrast to client-server set-ups cooperative overlays introduce the dynamic of participants joining, leaving and failing during the service, which can cause significant jitter and packet loss. Fortunately, multimedia streaming services can tolerate the loss of a certain fraction of packets. Only if a specific error rate is exceeded, the perceived quality of the presentation drops dramatically. Hence, a streaming system can be defined as working properly as long as all participants receive the stream in an acceptable quality. The damage in turn can be measured by the fraction of defective or missing frames at the receivers.

Overlay streaming systems basically consist of two services:

- a look-up service to locate resources like content and participants
- a streaming service for parent selection and data delivery

In this paper we are concerned with the nature of the streaming service. It has high data rates and is required to efficiently deliver the entire content with strict timing constraints to all participants of the system. The voluminous content can not be cached in large quantities nor can it easily be reproduced. An additional limitation is the fact that the content originates at a single source and is routed through the whole overlay. Every node consequently is dependent on the successful operation of all preceding forwarding nodes, which are part of the path between the source and itself. Every node failure initially leads to the loss of all packets it would have forwarded at all its child nodes and successors. As the bandwidth of participating nodes is limited, the nodes usually cannot maintain a high connectivity or simply switch source nodes. Furthermore, redundant delivery of packets immediately leads to unwanted traffic overhead. These characteristics have the effect, that failures, or even worse, intentional attacks on important nodes in the streaming overlay can have a high impact on the quality of the received service.

While fundamental work has been done to understand fault tolerance of networks [1] and it is comparably easy to create topologies which are resistant

to random node failure, accomplishing attack resilience seems a significantly harder task. By implementing quick fall-back strategies and creating topologies with many leaf nodes and a small number of forwarding nodes only, high resilience towards node failure can be achieved. The load in this case is provided by a few nodes only and node failures in the large set of receiving nodes do not have any impact. An attacker however will most probably try to gain as much information about the topology as possible and choose an important node to break. After finding nodes which directly or indirectly serve high amounts of other nodes through protocol analysis or probing, a few successful attacks can lead to the complete destruction of the service.

Consequently, in addition to building efficient overlays a main goal has to be the robustness of the topologies, not only to node failures but more importantly to attacks. The algorithm has to construct topologies with evenly distributed dependencies in order to avoid large groups of nodes relying on a small set of predecessors only. The streaming trees need to be as low as possible as unnecessary dependencies are introduced otherwise. In addition the algorithm needs to suffice with little information only, to make it impossible for an attacker to quickly detect important nodes as targets. Construction of good topologies and keeping as much information secret as possible are obviously competing goals. Hence a good trade-off has to be found.

In this paper we present a new scalable distributed algorithm which creates attack robust overlay streaming topologies. Our aim is to achieve resistance by creating topologies with many short source-receiver paths. In order to obtain a low degree of dependency between any two connected nodes, the content is striped into equally sized parts. Each node chooses different parents for every stripe to confine the damage of a single failing parent to the respective part of the stream. Short paths between the source and all participants are constructed to avoid chains with some nodes being dependent on many preceding nodes in the concerned stripe. As a result, the created overlays have a high node connectivity and low diameter.

The rest of the paper is organized as follows: we present the related work in section two, followed by an introduction of a model and of our approach in section three, and a simulation study in section four, before we draw a conclusion and describe further work in section five.

2 Related Work

Research on resistant and stable peer-to-peer streaming systems can be divided into three main categories. The first approach is to achieve a good quality of the stream at the receiver through means of FEC-like content encoding schemes [8,11]. A second class of systems [2,10] builds overlays with path diverse routes on the network layer. The third approach, including PRM[4], FatNemo[5], DagStream[9] and SplitStream[6], attempt to construct overlay topologies, which are robust to errors and attacks.

Probabilistic Resilient Multicast (PRM)[4] is a derivative of the NICE[3] application layer multicast which in addition to the regular packet relaying implements a randomized forwarding scheme and retransmission of lost packets. Every node on top of the usual child nodes, which are provided with every received packet, chooses some additional nodes at random. Packets are forwarded to the additional nodes with a low probability, thus introducing a slight redundancy in the data delivery which leads to a lower susceptibility to node failure. Similar to PRM, FatNemo[5] is a hierarchical clustered application layer multicast derived from NICE. While in NICE every cluster has one cluster leader, which is responsible for managing the groups and forwarding the packets to all siblings in each cluster it is a member of, FatNemo introduces Co-Leaders and caching strategies in order to increase the resilience. It additionally organizes the nodes into "Fat Trees", with broadband nodes being moved close to the source and low bandwidth nodes moved to the edge of the system. Both PRM and FatNemo do not achieve robustness by specifically constructing stable graphs but by redundancy, reordering and caching.

DagStream [9] attempts to create resistant topologies through increasing the vertex-connectivity of the streaming overlay. Alongside the main goal of creating a stable topology the second goal of DagStream is to implement network awareness in order to achieve network efficient overlays. The approach of DagStream is to organise peers into a directed acyclic graph (DAG) with each node connecting to a minimum number of parent nodes. Hence, some of the main issues of the system are locating source nodes and keeping the system circle-free. In previous work [12] we have considered constructing a DAG as the routing topology for a live overlay streaming system as well. However, our results showed, that the vertex-connectivity does not increase together with the amount of selected source peers for each node. In fact, our experience is that it is close to impossible to maintain high vertex connectivity as a few nodes at some level of the DAG serve many other nodes and act as very relevant hubs. They become a minimum cut set and their failure has serious impact on the performance of the system. Another problem we identified is the need of information about the distance of a node to the stream source. It on the one hand is necessary for creating loop free DAGs, but gives a possible attacker easy access to knowledge about the topology on the other.

The main goal of SplitStream[6] is to balance the load evenly between all participating nodes and to avoid an unfair distribution of the load between a small subset of forwarding nodes versus many leaf-nodes in the multicast tree. In order to create node-disjoint paths for all stripes, each node aims at forwarding only the stripe with an id-prefix matching its own. The forwarding topology is created as a RPF route of the joining request, with all peers en route until the first node, which already is a member of the group, is reached. To transfer different stripes (parts of the stream), each stripe is registered as a unique multicast group. SplitStream introduces a spare capacity group of which each node with available bandwidth is a member, to avoid bandwidth

shortages and dead locks. Its design of interior-node disjoint trees for every stripe of a layered streaming codec leads to a better stability against random node failure, as long as not too many nodes are served through the spare capacity group. Regarding the resilience, SplitStream has two shortcomings. The first is the fact that SplitStream does not consider the height of the branches in the streaming trees, thus under unfortunate circumstances it is more susceptible to node failures. The second and more important problem is the fact, that in SplitStream every node can gather information about the topology and especially locate the first nodes in each stripe, as groups and stripes are mapped to the node-ids.

3 Resilient Streaming overlays

With the content being routed from the source through the whole system, ALM create topologies of connections between neighboring nodes. Single source streaming systems build chains or multicast trees and multi-source streaming systems create a DAG or cyclic network of disjoint multicast trees. These topologies can be represented as special instances of graphs and used as a system model.

3.1 Model

In principle one participant, v_s, is the original source of the multimedia content (the camera together with the attached networking device). After joining the system a requesting node v_r locates potential parent nodes, which have already joined the streaming service, as forwarding relays. Every participating node in turn for receiving the content offers the service of relaying it.

Consequently, an overlay can be modelled as an undirected graph $G = (V, E)$ with a finite set of vertices $V = \{v_1, \ldots, v_n\}$, a data source $v_s \in V$ and the set of edges: $E \subseteq \{(u, v) | u, v \in V, u \neq v\}$. An additional stream $\mathcal{S} = \{p_1, \ldots, p_p\}$ of p packets, which can be replicated at each vertex, originates at the data source. The packet stream alternatively can be split into partial streams, with l sequences of k stripes: $\mathcal{S} = \{\{p_1^1, \ldots, p_k^1\}, \ldots, \{p_1^l, \ldots, p_k^l\}\}$

Neighbor selection follows local routing decisions which are constricted by the limited available bandwidth of the relaying nodes. All decisions should lead to an efficient mapping of the overlay to the underlying backbone to avoid unnecessary traffic. The aggregated routing decisions lead to the topology of the system.

To model an efficient routing, let

(i) $d : E \to \mathbb{R}^+$ be a non-negative edge-length (the latency of the connection)

(ii) $c : V \to \mathbb{R}^+$ be the vertex-capacity (bandwidth of the access-link)

The problem is to find k rooted spanning trees: $T_1 = (V, E_1), \ldots, T_k = (V, E_k)$

in G with minimum total cost

$$\sum_{i=1}^{k} d(T_i) = \sum_{i=1}^{k} \sum_{e \in E_i} d(e),$$

constrained by the degree of each vertex $v \in V$ in all T_i being at most $c(v)$:

$$\sum_{i=1}^{k} \deg_{T_i}(v) \leq c(v) \quad \text{for all } v \in V$$

with v_s being the root in all trees. Disjointly merging the spanning trees results in the multigraph $D = (V, E_1 \sqcup \cdots \sqcup E_k)$ of the streaming topology.

Some basic stability characteristics can already be derived through analyzing the properties of this multigraph D. If the source splits the content into k stripes and has a bandwidth capacity of delivering the whole content C times, $C \cdot k$ nodes at the most have a direct connection to the source. As the breakdown of a node leads to packet loss at all of its successors until the topology is mended, a topology can be destroyed completely by simply removing all $C \cdot k$ nodes which are connected to the source. Broadening the topology below, these first nodes remain the minimum cut set. Due to inappropriate routing decisions however, the topology can develop hourglass characteristics, with a smaller number of nodes becoming the minimum cut set. Because of the high node dynamic in overlay streaming systems and the dependence on all predecessing nodes, the path between the source and all receiving nodes has to be as short as possible.

Multi-source approaches, which connect child nodes to more than one relaying parent, implicitly create a multitude of trees. If the resulting topology is a directed acyclic graph or a network with cycles depends on the routing decisions of the nodes. The construction of directed acyclic graphs and networks still hold different possibilities to achieve stability. While in a network a node in different stripes can be be both parent and child of another node, in a DAG all nodes are organized strictly hierarchical. In the event of a failing node, the orphaned child nodes in a DAG need to find out the position of other nodes in the hierarchy, in order to locate alternative parent nodes and to avoid loops. Thus, systems creating a DAG have the two draw-backs of an uneven relevance distribution of the nodes and the possibility to gain knowledge about their importance. In networks, the importance of nodes can easier be balanced by rearranging them.

The most promising approach to create stable topologies is to keep all trees balanced and as low as possible and, like in SplitStream [6], to organise all participants into trees, with every node being a relaying peer in one tree only and a receiving leaf in all others. The latter strategy has the effect that the node connectivity does not decrease below the amount of stripes but the resulting digraph broadens in all branches. In this best case, the minimum cut

set is the whole set of nodes, which are directly connected to the data source.

3.2 Basic Protocol description

After introducing the properties of potential streaming topologies, we want to present our approach of creating stable topologies. The original stream is divided into k stripes and forwarded on disjoint paths by means of each node forwarding only one of the stripes. This stripe selection is done by each node choosing the stripe it forwards to the most other nodes. The streaming service consists of four operations: `connect`, `disconnect`, `data delivery` and `pass child`.

3.2.1 Joining

The operation of joining a stream is divided into two basic phases. At first a stream and participating nodes are discovered and the streaming service is joined afterwards.

To locate streams of different content and in order to be able to find a first set of potential parent nodes, a look-up service is needed. In order to construct a network-efficient overlay, a topology aware peer-to-peer look-up service is used which is implemented as a distributed hash table (DHT) with location based node-id assignment. As soon as the node knows about potential parent nodes, it chooses the one with the closest id and issues a `connect_request` for all stripes. The contacted node then has to either send a connection confirmation and start forwarding packets or reply with a `disconnect_request`. Both the `connect_confirmation` and the `disconnect_request` primitive contain an alternative relaying peer of the requested stream. If a node receives a `disconnect_request` from the chosen parent, it sends a new `connect_request` to the alternative node from the disconnection message.

3.2.2 Topology Control

On reception of an incoming connection request a node has to accept the new connection and establish a link to forward the packets of the requested stripe to the new child node, as long as it has available bandwidth left.

If a requested node runs out of available bandwidth and is not able to serve any more child nodes, if a node detects major changes of the topology below, or if a node is actually requested to forward a stripe other than its preferred stripe, it initiates a local topology optimization. It either disconnects one or more of its links and moves child nodes to different relaying nodes or asks one of its child nodes to hand over a part of the nodes they serve to re-balance the amount successors below.

The topology optimization is done in two phases. In the first phase the node compares all of its child nodes and determines which node poses the highest cost and adds the least benefit to an optimal topology. Cost is measured through evaluating three factors in the following order: the number of

links a node maintains to the child, if the child receives a stripe different from the preferred and if the child itself serves only few other nodes. In the second phase the node from all its childs, which forward the disconnected stripe, selects the node which serves the least other nodes as an alternative source and sends a disconnect notification to the dropped node.

Strict division of stripes in some cases can lead to dead locks. In this case a node can not drop all expensive links and has to forward another stripe in addition to its preferred one. However, it will try to pass the respected link to a different node in the next round.

In order to be able to keep the topology balanced, forwarding nodes keep track of the amount of successors of all its child nodes and their successors. Every node updates the successor information with its parents when it observes a situation change. To keep the protocol overhead low even in highly dynamic phases, this is done after waiting $10ms$ for additional changes.

If a forwarding node detects an uneven topology, which it cannot solve by moving a node to another one of its child nodes, or if a parent due to node-leaves again has some bandwidth available, it sends a `pass-node_request` to its child with the most successors. The child node then sends a `disconnect_request` to one or more of its childs with its parent as the alternative node.

3.2.3 Leaving

Node leaves may happen for the two different reasons of gracefully quitting a service or node failure. In the event of a node deliberately leaving the service it issues a `disconnect_request` together with the parent, it itself receives the forwarded stripe from, to all child nodes. A node failure at a parent is detected through a timeout and reacted upon by connecting to the alternative node which the parent sent together with the connection confirmation. If this fall-back node is no longer part of the system, a connection request is sent to one of the other existing parents. In case that none of the old parents nor their predecessors remain alive in the system, a node has to start searching parents using the look-up service again.

4 Performance Study

In order to examine our algorithm and the routing decisions it takes, we evaluated the characteristics of the evolving topologies. Simulations of the system were conducted using OMNeT++, with varying capacities and amounts of stripes at first, to acquire snapshots of the topologies for further testing. To judge the stability, we used two metrics to measure the resistance: the mean node connectivity and the fraction of received packets after node removal.

Node connectivity gives a good indication as to how many nodes have to be successfully attacked in order to achieve a fragmentation, completely disconnecting a group of nodes from the streaming overlay. It should equal

the number of stripes, as every node tries to forward its preferred stripe only, for all bandwidth capacities.

As the service for a receiver already has to be rendered useless as soon as a comparably low percentage of packets are missing, we define the service as working properly as long as at least a certain proportion of the correct packets can be received. To analyze the resilience to this service breakdown we implemented graph representations of the topologies and measured the impact of node disconnections to the amount of receivable packets. The nodes to be disconnected were chosen randomly as well as following a perfect (greedy with global knowledge) and a realistic attack (weighted selection). Again, in stable topologies, node removal should lead to the packet loss of only one stripe at the succeeding nodes only, keeping the service alive while decreasing the quality.

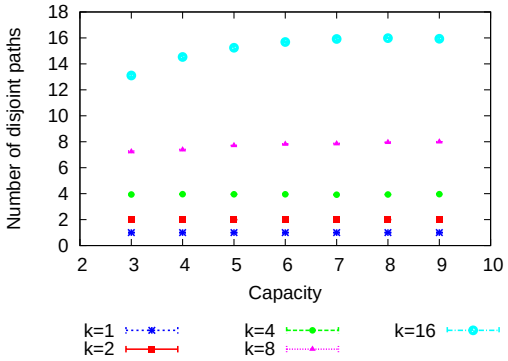

Fig. 1. Mean amount of disjoint paths to the source (V=500)

The study did not always yield the expected results. While for lower numbers of stripes, the algorithm created an almost equal amount of node disjoint paths, we observed that the gain in the node connectivity did not grow as much as anticipated. For rising amounts of stripes with low bandwidth capacities, the number of node disjoint paths increased contractively linear (with a gradient below 1) only (See: Fig. 1). In topologies which were created streaming 16 stripes, a low bandwidth capacity obviously led to an hourglass characteristic with a significant amount of nodes being served through less than 16 paths. Increasing the capacity helped solving the bottleneck situations and the topologies showed an asymptotic behavior to the theoretical limit of 16 disjoint paths. Additionally, increasing the number of stripes at any capacity always led to a higher vertex connectivity, indicating that the topologies were more stable the higher an amount of stripes was streamed.

In the second step, we exposed the same topology snapshots to node disconnections. The complete routing topologies were represented as a forest of multicast trees and nodes removed until less than 50% of the original streaming packets were received correctly: Given the directed multigraph D of the topology snapshot and the functions:

$$\mathrm{inc}_{\mathcal{A}}(v) = \text{ In-degree of } v \text{ in } D \setminus \mathcal{A}$$

$$\mathrm{drop}(\mathcal{A}) = \sum_{v \in V \setminus \{s\}} (k - \mathrm{inc}_{\mathcal{A}}(v))$$

we tested, how many nodes $\mathcal{A} \subset G \setminus \{s\}$ had to fail, with node failure causing the removal of all succeeding links of the failing node, leading to $drop(\mathcal{A}) \geq r_d \cdot k \cdot V$ with $r_d = 0.5$. This threshold of 50% is merely theoretic, as the quality is highly dependent on the used codec and the amount of remaining participants on the requirements of the service provider.

To get an idea of the worst case stability we tried to find the minimum set $\mathcal{A}$ and implemented an attack based on global knowledge at first. For greedy selection, the successors of all nodes were calculated and the node with the highest successor count disconnected in each round. Under greedy attack, in a completely balanced forest with inner node disjoint trees, the topologies should withstand the disconnection of $\frac{C \cdot k}{2}$ of the nodes which are served by the source directly. They have developed an hourglass characteristic otherwise, leading to an unstable routing. For all bandwidth capacities and up to four stripes, the topologies showed a behavior which was close to the anticipated results (See: Fig. 2). However, further increasing the stripes to 8 and 16 again led to a contractively linear gain only.

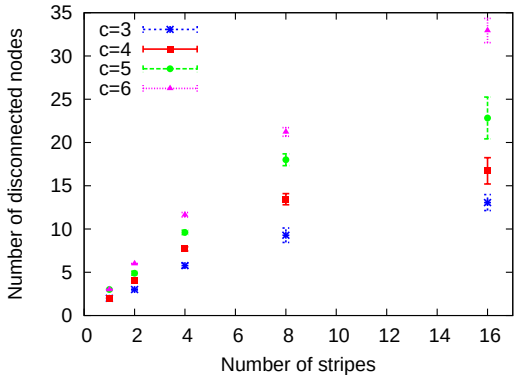

Fig. 2. Greedy removal decreasing received packets to 50% (95% confidence, V=500)

In order to examine the resilience of the topologies in more realistic scenar-

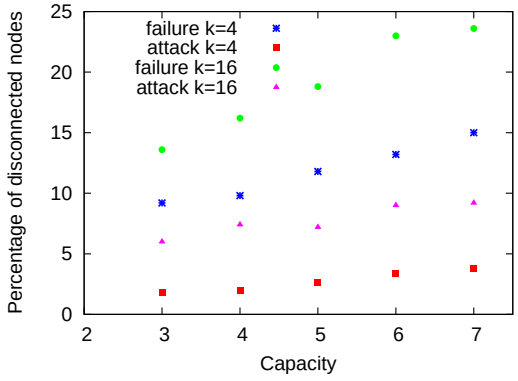

Fig. 3. Worst case resilience at random node failure and realistic attack (% disconnected nodes, V=500)

ios, we additionally implemented a realistic attack and random node deletion. Usually, an attacker will try to gain knowledge about a system, before choosing a node to attack. As it is impossible to probe the complete system due to the protocol, an attacker can only collect knowledge about the topology below. Consequently an attacker could only try to become as important as possible and, at a certain point, to stop the service. Or the attacker could by chance get to know an important node, when receiving a connection request, and attack this node in turn. In order to simulate such a behavior, we implemented a weighted failure of nodes. Each node in every round had a failure probability proportional to the amount of its successors and nodes were disconnected, until the threshold of lost packets was reached. Random node deletion, as a third test, resembles the failure of nodes. It causes very widespread results, but gives a good estimation as to how many nodes need to fail at the same time, before the service breaks down. The results of both the attack and the failure (See: Fig. 3) again indicated contractively linear gain in stability with growing capacities and numbers of stripes. In a system streaming 4 stripes, an attacker would have to get to know around 1% of all nodes and disconnect them at the same time, to cause a denial of service. This amount rises to 2.5% with increasing capacities. Streaming 16 stripes it even rises to 6% of the nodes at a node capacity of 3, again rising to 8.5% when the node capacity increases to 6. Even higher numbers were measured for the random node failure: at the comparably low capacity of 3, with only 4 stripes 9%, and with 16 stripes 13.5% of the nodes had to fail at the same time, to cause the service to fail. These numbers again constantly rise with increasing node capacities.

To discover, why the evaluation did not always lead to the expected gains, we analyzed the topologies, which did not achieve the estimated results. Here we had to realize, that the gaps between the amount of additional stripes and both the rising stability and vertex connectivity were actually caused by deadlock situations. In the topology construction, large subsets of nodes selected the same preferred stripe. This behavior led to problems that some nodes kept forwarding more than one stripe and that links of some stripes were always dropped further down in the topology, thus leading to long paths.

5 Conclusion and Further Work

In this paper, we presented a novel distributed algorithm which, based on local knowledge only, creates network efficient and stable streaming overlays. We have shown that resilient streaming topologies can be constructed by following three strategies: Striping the content and (a) forwarding all stripes on inner node disjoint trees, which (b) have to be as flat as possible and balanced by (c) levelling the amount of successors of each node. In order to create attack resistant topologies, information about the parents and preceding nodes needs to be disclosed and is generally not needed.

Currently, we are examining the behavior of the dynamic system under attack. We are thus exposing the dynamic simulations of the system to greedy attacks based on global knowledge in order to measure the volatility and messaging overhead as well as the amount of lost packets due to different amounts of successfully attacked nodes. In future we plan to modify the algorithm to avoid the detected deadlocks in order to prevent long paths to evolve.

References

[1] Albert, R. and A. Barabasi, *Statistical mechanics of complex networks*, Reviews on Modern Physics **74** (2002).

[2] Andersen, D., H. Balakrishnan, F. Kaashoek and R. Morris, "Resilient overlay networks," ACM Press New York, NY, USA, 2001.

[3] Banerjee, S., B. Bhattacharjee and C. Kommareddy, *Scalable application layer multicast*, in: *ACM Computer Communication Review*, 2002, pp. 205–217.

[4] Banerjee, S., S. Lee, B. Bhattacharjee and A. Srinivasan, *Resilient multicast using overlays*, in: *ACM Performance Evaluation Review*, 2003, pp. 102–113.

[5] Birrer, S., D. Lu, F. Bustamante, Y. Qiao and P. Dinda, *FatNemo: Building a resilient multi-source multicast fattree*, in: *Proceedings of the Workshop on Web Content Caching and Distribution*, 2004, pp. 182–196.

[6] Castro, M., P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron and A. Singh, *SplitStream: High-bandwidth content distribution in a cooperative environment*, in: *Proceedings of (IPTPS'03)*, 2003, pp. 298–313.

[7] Chu, Y.-H., S. G. Rao and H. Zhang, *A case for end system multicast*, in: *Measurement and Modeling of Computer Systems*, 2000, pp. 1–12.

[8] Goyal, V. K., *Multiple Description Coding: Compression Meets the Network*, IEEE Signal Processing Magazine **18** (2001), pp. 74–93.

[9] Liang, J. and K. Nahrstedt, *DagStream: Locality Aware and Failure Resilient Peer-to-Peer Streaming*, in: *Multimedia Computing and Networking (MMCN)*, Proceedings of SPIE **6071**, 2006, to appear.

[10] Nguyen, T. and A. Zakhor, *Distributed video streaming over Internet*, in: *Proceedings of SPIE*, 2003, pp. 186–195.

[11] Padmanabhan, V., H. Wang, P. Chou and K. Sripanidkulchai, *Distributing streaming media content using cooperative networking*, in: *Proceedings of ACM/IEEE NOSSDAV*, 2002, pp. 177–186.

[12] Strufe, T., G. Schäfer and A. Chang, *BCBS: An Efficient Load Balancing Strategy for Cooperative Overlay Live-Streaming*, in: *Proceedings of The IEEE International Congress on Communications (ICC)*, 2006, pp. 304–309.

[13] Wong, T. and R. Katz, *Analysis of multicast forwarding state scalability*, 2000 International Conference on Network Protocols (2000), pp. 105–115.