Towards the design of unexploitable construction mechanisms for multiple-tree based P2P streaming systems

Michael Brinkmeier[2], Mathias Fischer[1], Sascha Grau[2], and Guenter Schaefer[1]

[1] Telematics and Computer Networks, TU Ilmenau
[2] Automata and Formal Languages, TU Ilmenau

Abstract. In peer-to-peer based live streaming systems, a great number of participants have to cooperate to efficiently and reliably distribute a continuous flow of data. Each receiving peer in return provides its resources to the system. Since these systems operate in a completely distributed manner, it is of particular importance, to prevent malicious members from harvesting important topology information or influencing the streaming system to their needs. In this article, we analyze potential attack methods on multiple-tree-based P2P streaming systems, discuss important design decisions to constrain the impact of malicious behaviour, and we introduce the new concept of peer testaments. By analyzing existing systems, we show that so far only few attention has been given to the design of unexploitable construction mechanisms. Based on the identified design decisions, we propose a novel streaming system and evaluate it by exposing it to different types of internal attackers. Our results show that these attackers have to spend large effort to reach relevant positions in the streaming topology and that their bandwidth contribution far outnumbers the damage they achieve.

1 Introduction

In recent years, peer-to-peer (P2P) based Application Layer Multicast (ALM) live streaming systems have become a major topic of interest, both in research and practical application. In such a system, a continuous flow of data - the stream - is distributed from a source node to a great number of peers by utilizing their resources for redistributing the stream to other peers. A major advantage of such a design is that the number of participants can grow independently of the source's upload bandwidth.

ALM systems are commonly classified into pull and push approaches. Push approaches create and maintain an explicit topology for content distribution, whereas in pull approaches every node explicitly requests each part of the stream at other participating nodes. So, pull approaches require to preload the stream well in advance of the playout, which causes relatively high delays. Therefore, they are out of scope of this article and the main emphasis is put on push-based approaches, which organize their participants in a *topology* of one or more spanning trees, each rooted at the source node. If multiple trees are used, the source

divides the stream into equally sized substreams called *stripes* and distributes each one via its own tree, thereby removing bottlenecks and improving resilience. Thus, by using Multiple Description Coding or Forward Error Correction methods, it is possible to compensate the loss of one or several stripes.

Consequently, each peer takes part in every tree and automatically receives each stripe from a node one step closer to the source, which is called *parent* or *predecessor*. Equivalently, it redistributes one or more stripes to its *children*. The set of all nodes transitively receiving data from a peer is its *successor set*. Generally, we will say that a node is on a *low level* in a certain tree, if it is located far away from the source.

In order to construct and maintain such topologies, a distributed mechanism is required, especially since the set of participants is highly dynamic, because of joining and leaving nodes (the so-called *node churn*). Elementary operations of such a mechanism are *bootstrapping*, that is the way nodes join the system, and *repairing* of trees after the announced or unannounced exit of nodes. Furthermore, multiple kinds of optimizations like balancing and flattening of the trees, as well as bandwidth and timing optimizations are commonly implemented.

All these actions are performed by the participants using a common strategy. Since topology maintenance always influences the position and hence the size of a node's successor set, it is important to design all mechanisms in such a way, that they cannot be exploited by malicious participants for conducting attacks against the distribution mechanism.

In a first step, this article collects basic properties of attacks on push-based streaming systems in section 2. Our main contribution is given in section 3 by discussing building blocks for unexploitable construction mechanisms for push-based ALM streaming systems, including the new concept of *peer testaments*. Current streaming systems are classified and analyzed in section 4 according to the proposed building blocks, revealing that most of them provide no effective defense against internal attackers. So, in section 5 we propose a streaming system that includes an unexploitable topology construction mechanism, which gets experimentally analyzed in section 6. Section 7 summarizes our findings and gives an outlook on future work.

2 Attacks against P2P streaming topologies

There are two basic types of attackers, the *external* and the *internal* one. An *external attacker* can only observe and attack the system from outside. Thus, it may issue attacks on participating nodes without being sanctioned by the system, allowing aggressive approaches. In general, we assume, that an external attacker can take down a specific node by issuing a DOS attack against it, or triggering some other effect which removes it from the system. The main limitation of an external attacker is the fact, that it has to gather information by eavesdropping the communications between nodes, which can be restricted by using cryptographic protocols.

Unless an external attacker can gather a lot of information about the system, an *internal* attacker is more powerful, especially if combined with an external component. In the following, we specifically concentrate on variations of *sleeper attacks* with additional capabilities, covering a wide range of possible attack scenarios, since many attacker models can be reduced to a sleeper attack.

The *sleeper attack* is one of the most basic attacks. The attacker controls a set of *agents*, which participate in the system. In addition, they may communicate with each other or with a central authority, guiding the attack. The agents join the system in the same way ordinary nodes do, and start participating. In this way, they can easily use all procedures and protocols that the system provides.

In a *passive* sleeper attack, these agents simply wait until a certain condition is met and then an attack is triggered. The point in time at which this happens, may be chosen in many ways. Either the agents attempt to reach certain relevant positions in the topology, or they just collect information and try to identify important nodes or even the (possibly concealed) source itself. The second aim is basically equivalent to the first, since usually the source can be identified by a node that reaches a specific position. Hence we may assume, that an attack is triggered as soon as the positions of the agents satisfy specific conditions.

How the agents (or the controlling authority) decide whether their current positions are 'important', mainly depends on the signalling procedure of the particular P2P streaming approach. If, for example, the system provides its participants with informations about the number of other nodes depending on their service (ie. their number of successors) or even the IDs of these nodes, the attacker can calculate the exact damage it can cause at any given time. Hence, it may simply trigger the attack, as soon as a given bound is reached. In other situations, the position of a node may correlate with the time it was in the system. In these cases, the attacker may simply wait for a specific time to pass and then has a good chance that his agents may cause a given damage.

If the attacker decides or assumes, that the agents have reached good positions, it can trigger an attack. This might be a simple disruption, caused by all agents leaving the system at the same time. Optionally, the agents may not leave the system completely, but reduce their bandwidth, such that the quality of service for nodes depending on them decreases significantly. Another way to disrupt the service is a coordinated attack on specific nodes, which were previously identified by the agents. These attacks can either be issued by the agents themselves, risking their discovery and possible sanctions like exclusion, or by an assisting external attacker, leaving the agents concealed. Furthermore, the agents can provoke damage with a *resource-consuming attack*, during which a large number of agents try to connect to the same node with the goal of preventing the target from forwarding the stream to other participants. This is a variation of a DOS attack, consuming the bandwidth of a specific node.

Alternatively, the agents may start to *pollute* the system by inserting corrupted data into the stream. Successors may detect the pollution (e.g. by verifying cryptographic signatures on the stream data) and try to reconnect to the system, possibly leading to a massive reconstruction and a decrease in the qual-

ity of service. This *Pollution attack* can have the same effects as the failure of the agents or the attack on specific nodes.

In addition to simply cooperating with the streaming system, the agents may pursue their task to reach relevant positions more *actively*. For example, in surprisingly many streaming systems, peers can promote themselves to more important positions. They are often even rewarded for taking on more responsibility, with the goal to inhibit freeriders.

A second way for the agents to gain importance is to lie about their performance, making them seemingly better than other peers. This misinformation may lead to a 'false promotion', since the other nodes assume, that the agent is the best possible choice for an important position. Or they may try to attract peers to become their children, increasing their number of successors. Third, they might support each other, eg. by recommending other agents for promotion or giving them high reputations. Fourth, they can become aggressive in advance by attacking other peers in their surroundings, provoking an (at least local) reconstruction of the topology, thereby giving them the chance of reaching a better position. As above, these supporting attacks might be conducted by an external assistant, preventing the discovery of the agents.

One important parameter of sleeper attacks is the degree of cooperation between agents. The agents may be completely independent from each other, preventing coordinated attacks – except if a global synchronized clock is used. Or they may communicate via an external authority, allowing a coordinated attack and the simultaneous collection of information from all agents. The costs and modalities of communicating with the coordinating authority can be used to model some specific types of attacks. For example the Sybil attack [Dou02], in which agents are logical instances of the same physical node, may be modeled by assuming, that the coordinator has at every time the knowledge of every agent.

In contrast to an external attack, an internal attacker supports the attacked system for a while. Hence, the *efficiency* of an internal attack can be measured by the quotient of the caused damage and the invested cost. Let X be a set of nodes, which we assume to be agents or attacked due to the information, the agents gathered. An estimate for the damage caused by the attack at time T is the number of successors of the nodes in X at time T summed up over all stripes, i.e. $\text{damage}(X,T) = \sum_{i=1}^{k} \text{succ}_{i,T}(X)$, where $\text{succ}_{i,T}(X)$ is the number of nodes that are successor of at least one vertex in X in stripe i at time T. The cost invested by the attacker can be measured by the total bandwidth, that his agents provided over the time, i.e. $\text{bandwidth}(X,T) = \sum_{t=1}^{T} \sum_{v \in X} \text{fanout}(v,t)$, where $\text{fanout}(v,t)$ is the bandwidth that vertex v provided at time t. The *efficiency* of an internal attack if given by the quotient of the caused damage and the invested bandwidth, ie. $\text{efficiency}(X) = \frac{\text{damage}(X)}{\text{bandwidth}(X)}$. Using this measure, the vulnerability of different systems against a specific attack can be compared, with a higher value indicating a lower resilience. Alternatively, different attack strategies on the same system can be compared.

3 Requirements for manipulation-proof construction mechanisms

The basic principle of manipulation secure topology management mechanisms is *mistrust* in all participating nodes. In such a mistrustful system, every node is suspected to be malicious, is therefore provided with the minimum required topology information and has to proof its reliability, e.g. by long-term cooperation, before being considered for taking over responsibility.

When implementing this idea in a functional P2P streaming system, a list of requirements for key design decisions arises, that concern the direction of the distribution of topology information as well as repair mechanisms, choosing the right nodes for node promotion, and bootstrapping issues.

3.1 Distribution of topology information.

To build both efficient and reliable streaming topologies, it is necessary to make a certain degree of topology information available to the participating nodes. Examples for such information reach from the numbers of stripes/nodes/successors over the IDs of nodes in relevant positions to a complete snapshot of the current topology. Of course, it would also be possible to build topologies without any more information than the IDs of nodes in a local neighborhood, however the resulting topologies can become rather inefficient or unstable.

On the one hand, there is a trade-off between supporting topology management decisions with more information, and on the other hand providing malicious agents with the same data, which allows more effective attacks. Hence, all information distributed should at least have a local character, such that it is related only to limited regions of the topology and is only available to nodes from such regions. A typical example is the successor number of a node and its children, allowing important tree balancing operations. Without an estimate on the total node number, a prediction of the own importance in a topology is not possible.

In addition to the kind of information to be distributed, the direction of information flow is equally important. Clearly, a concentration of knowledge near to the source is advantageous compared to a distribution towards the tree leafs. This way the number of involved nodes is minimized and their reliability must have been previously certified in some way. A related rule in the design process must be the strict separation of stripe-related information. So, data describing the role of a node in the distribution tree of a certain stripe should not allow conclusions about its role in other stripe trees. This is necessary, since a node being a predecessor in one stripe and hence receiving such information, may be on very low levels in other stripes. Thus, without a separation, the source-directed information flow would be violated.

3.2 Locality of repair mechanisms.

As soon as the failure of a relaying node is detected, appropriate topology repair mechanisms have to be performed. Especially, a substitute node from a lower

level has to be found. Here, we can again choose between different alternatives. At first, a repair mechanism considering nodes from the whole topology seems to be desirable. This way, a node from a completely different subtree could take over the role of the failed one. Thereby, the node profiting from the failure would not belong to the direct neighborhood of the failed node, thus making attacks on direct predecessors worthless.

However, without a coordinator with global topology knowledge, such a system has a number of major drawbacks. First, the substitute node has to be chosen, possibly involving a high number of other nodes, each of questionable trust. Additionally, due to long communication paths, such a mechanism will imply long repair times which are furthermore dependent on the total number of topology nodes (the time may grow at least logarithmically with the number of nodes). At last, the substitute node has to be replaced in its old position, too, raising questions of convergence and the undesirable property that local failures propagate through the whole topology.

Therefore, a local repair method in which the substitute node is chosen among the children of the failed one, is preferable. This way restricts the repairs to the subtree rooted at the failed node. Furthermore, the risk that disabling a predecessor gives a chance to advance in the topology can partially be mitigated by one of the reliability estimation mechanisms described later.

3.3 Initiative of repair actions.

The first nodes noticing the failure of a node v, are its direct children and its predecessor (the exact order depends on the streaming system). Which of them initiates the repair is another important design decision. In many systems the initiative lies at the children; they either exit and rejoin the system or they try to attach to their grandfather. The first variant has to be discouraged, since it leads to unbalanced trees and requires a number of global rebalancing operations.

The second variant depends on regular updates on the grandfather's ID from each parent to its children. Clearly, this reveals a lot of topology information to possibly unreliable nodes. Furthermore, during the reconnection process the grandfather has to decide which of its affected grandchildren shall now take over the role as head of the complete subtree that was formerly lead by the failed node. Since the grandfather has no knowledge about the number and quality of its grandchildren as well as to shorten repair times, it will be tempted to choose the first candidate arriving. Clearly, an attacker disabling its own predecessor has a timing advantage over its siblings.

Thus, regularly distributing child information to the predecessor and assigning repair initiative to it, seems more appropriate. In this case, nodes detecting a parent failure have to wait for a certain time threshold before acting on themselves. Meanwhile, their grandfather chooses a new child node from its grandchildren based on the information formerly sent by the failed node (additionally ignoring connection requests of its grandchildren). When communicating this decision to the involved nodes, the grandparent should authenticate itself, e.g. by using a secret of the dead parent, which has been sent upwards before. Of course,

rebalancing and capacity-based adjustments of the topology are still necessary. However, they will be restricted to the subtree in which the damage occured.

3.4 Decision process for node promotion and degradation.

Whenever higher topology positions shall be assigned or nodes are chosen to be dropped to lower levels, the decision process is based on different properties of the available candidates. Once again, a trade-off between efficiency and reliability may occur. Typically compared key factors include available bandwidth, successor number, participation time and the effort a node has already invested in stream distribution. If available, the latter can be estimated by a trust or reputation system.

Independent from the decision which of these properties are considered, it is important to prevent candidates from lying, which requires some kind of proof. Depending on the checked property, this may be done by demonstration in response to some kind of challenge. Examples for such a procedure are bandwidth-tests or cryptographic puzzles for a successor estimation [RSS07]. Another kind of proof is to present witnesses, by direct communication or by forwarding cryptographically secured testimonies, e.g. certifying good long-term service.

An especially interesting kind of proof is a *testament*, namely a certification of a node about the suitability of its children as an heir to its position. In contrast to a grandparent, needing to make a fast decision during the repair process, such a testament is built over a long time of cooperation and may include many factors. Thus, it enforces exemplary behaviour to nodes willing to ascent in the topology. A simple way to use testaments, is by regularly submitting such a rating of the own children to the predecessor, providing it with far more consolidated information to choose its new child.

Note, that the chosen promoted node has no prior knowledge about its new children, which are its former siblings. To prevent a whitewashing, it is therefore important to initialize its own testament with the ratings made by the failed node. To do this, the grandparent has to forward the testament to the heir.

Another point to consider when designing the decision process, is its predictability. Clearly, a completely deterministic behaviour will be open to manipulation by well-informed attackers. Therefore, it is advised to add a random component, e.g. by equally choosing the node from some best portion of the candidates. A different modification would be to frequently choose the second-best candidate, thereby inhibiting lies about performance parameters, as long as attackers do not cooperate.

3.5 Reliability estimation via long-term service.

From the factors estimating the reliability of nodes, the participation time is one of the simplest. Furthermore, when statistically evaluating the distribution of participation times in live streaming systems [SM04], there is a high probability that old nodes will further continue to take part in the streaming system.

So, there is a concentration of long-living nodes on the high levels of the tree, the time between position changes grows. This effect automatically results from node churn under the precondition, that joining nodes are positioned as a leaf at lowest level (see later), but can also be actively encouraged by the design of the node promotion processes.

The ideal goal of such a mechanism must be to force the attacker to contribute so much exemplary effort to the streaming system, in order to reach an important position, that the resulting damage of its attack cannot compensate its costs. More specifically, the system has to be designed, such that the *efficiency* of an attack is as low as possible.

Additionally, information provided by a peer itself must always be mistrusted. A solution consists in measuring only the time spent in the current depth layer of the tree. This can be easily done by the peer's predecessor. Clearly, in case of a predecessor change, the new predecessor should initialize the time values based on the testament of the old predecessor. Due to its promotion, its own time value will be reset to zero.

3.6 Bootstrapping joining nodes.

A last important design decision concerns the way joining nodes are attached to the existing topology. This decision affects both, the availability of topology information and the balance of the tree topology. A very simple method is a node insertion at the source of the stream. This way, the new node can be dropped down to a position optimal for the tree balance. However, during this process it gains information about the source and nodes from all tree layers.

A more common approach is the use of bootstrapping servers returning random nodes as entry points. Since the great majority of nodes has only low positions and no information about their rank is provided, the information leakage is bounded. The similar variant, returning only random leaf nodes would require a significantly increased management effort and a single instance with global knowledge. Furthermore, it would not avoid necessary balancing and optimization operations, since without them the generated topologies grow very deep.

4 A Brief Discussion of Existing Systems

In some systems (e.g. HGM [RES01]) there exists a central coordinator, which has global knowledge and assigns participating nodes to specific positions. But by construction, such a coordinator is a single point of failure, which may be easily attacked, if not concealed. Furthermore, usually the existence of such a central authority limits the size of the system, preventing a good scaling behavior.

The well-known SplitStream protocol [CDK+03] requires that each node knows its path to the source in every stripe, in order to avoid cycles. Hence, every node knows the source. In addition the delegation of nodes is based on their ID and allows joining nodes with a suitable ID to take over relevant positions immediately. Since these IDs contain a randomized component, a new node

might be assigned to important positions purely by random, supporting an attacker with a large number of agents. Similar like SplitStream, some approaches like DagStream [LN06], rely on information flowing from nodes in higher levels of the tree to nodes residing in lower levels.

In approaches like Chunkyspread [VF06] or mTreebone [WXL07] the decision for node delegation and promotion is made by children nodes instead of their parents, which enables agents to lie.

The system proposed in [SWS07], operates with local topology knowledge and local repair mechanisms. The decision process for the promotion or degradation of a node is performed by its parent. However, this approach still has drawbacks. A child node does not only know its father, in addition it also obtains information about its grandfather, which is contacted by the child during a repair action.

5 A push-based streaming system utilizing a manipulation resistant topology repair mechanism

Our approach is built at the basis of [SWS07], as described in section 4, and extends it by utilizing the building blocks presented in section 3. Only local topology information is used and the information flow is strictly directed from leaf nodes upwards the tree, so that the view of participating nodes is restricted to their parents in each stripe, instead of their grandparents as in [SWS07]. So in our approach, a node communicates with its one-hop predecessor only, while it has broader knowledge about its children and the total number of nodes below, since this is required to enable a balancing of the underlying subtrees.

The main difference to [SWS07] is the use of testaments for node promotion during repair actions. In the process, a proper candidate for a promotion is chosen by the grandfather according to the node's residence time, measured locally at the father node. Therewith, the grandfather has the repair initiative and not the children of the failed node, as in [SWS07]. This combination of mechanisms now offers the evaluation of peers based on their long-term behaviour and avoids making decisions based on untrusted, snapshot-like information.

Bootstrapping of joining nodes is done by inserting them at random nodes that already take part in the streaming. A bootstrapping server provides a joining node with a list of parent candidates, which will either accept it as a new child or pass it on to one of their children.

6 Experiments

In order to evaluate our system with regard to the effectiveness of the integrated building blocks discussed in section 3, a simulation study was conducted. The system was simulated in the presence of local sleeper attackers with different capabilities. We measured the ratio between potential damage, represented by the overall number of successors of all attacking nodes, and the bandwidth spent until this time.

Within the simulation, the stream is divided into 4 stripes distributed via 4 spanning trees. Summed up over all trees, every client has enough upload capacity to supply at most 8 children. The source is capable of forwarding every stripe 3 times and therefore has a capacity of 12. A total number of 250 nodes was simulated, with 95% of them being ordinary streaming clients and the remaining 5% being sleeper attackers with different capabilities. Ordinary nodes join uniformly at the beginning of the simulation between the first 10 and 50 seconds and leave the stream following a shifted pareto distribution with $k = 1.3$ and $x_{min} = 1$. To keep the node number at a constant level, leaving nodes rejoin again uniformly distributed between 5 and 10 seconds after their leave. The overall simulation time comprises 300 seconds. Obviously, this user behavior may not reflect the reality, but it allows to observe the system at a high load and enables us to draw conclusions on the systems operation at a more realistic user behavior with less node churn and longer streams. Attacker nodes join slightly later to an almost complete but not fully established streaming topology, since we assume attacks on already running and stabilized streaming systems. It is clear, that during the unnatural massive node join at the beginning of the experiments, there is no way to prevent agents from being positioned at high levels. So, the attacker nodes join uniformly distributed between the first 30 and 60 seconds and remain in streaming until the end of simulation. 32 simulation runs were conducted per parameter set and for every graph 97.5% confidence intervals were computed.

Figure 1(a) shows the average residence time per level for all spanning trees, i.e. the average time that nodes in a certain distance from the source remained at this position. As discussed in section 3, a reliability estimation based on the participation time of nodes in the system, should allow to move more reliable nodes to higher levels of the tree. As can be seen in the figure, this is the case for our system, since nodes closer to the source have growing residence times.

In addition, we simulated sleeper attackers with different additional capabilities to test our topology construction mechanisms and to observe if an attacker can exploit them to ascent in the topology. In our experiments, the agents were able to kill their fathers once per a certain time interval. These varied between 0 (a passive sleeper) and 50 seconds. It is assumed that all agents were controlled by one authority, therefore their damage and invested bandwidths combine.

Figure 1(b) shows the results for the ratio of maximum potential damage per run to the invested bandwidth until that point in time. As expected, this value is increased by a high attack frequency. However, the attack efficiency is very low for all intervals, such that the attackers had to forward multiple times more packets than are now not able to reach their destination. Furthermore, the efficiency increase of the active attackers is low, demonstrating that the topology repair mechanisms successfully reduce the impact of such strategies.

Figure 1(c) shows the ratio of achievable damage to invested bandwidth (attack "efficiency", see also section 2) of the passive and the most effective active attacker over time. Once more, the active attack is more successful, but at a very low level. The efficiency of both follows a characteristic development. The highest benefit for an attacker lies between 40 and 50 seconds, shortly after join-

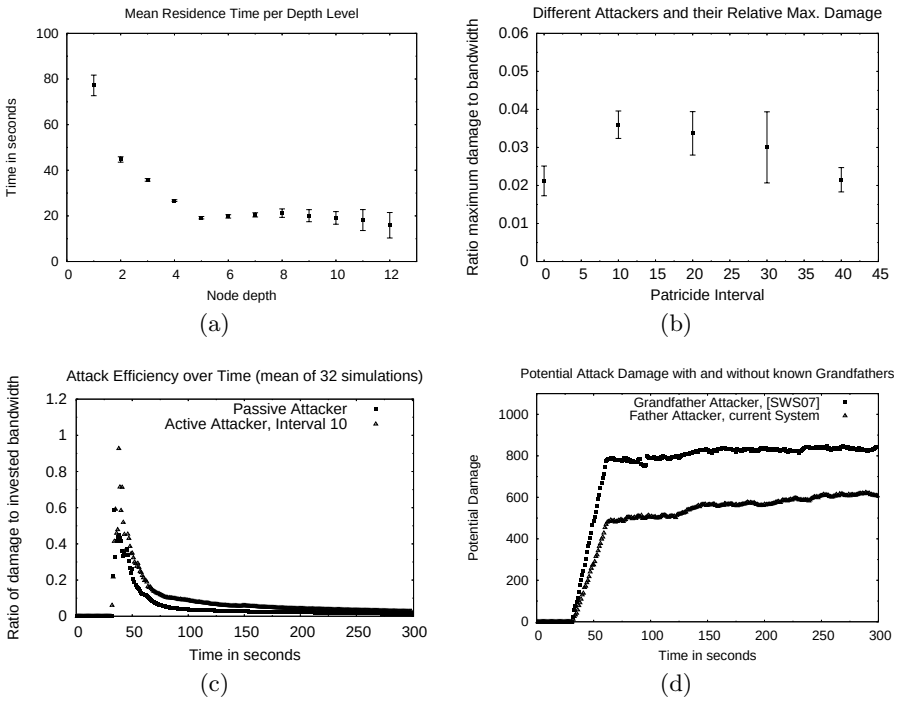

Fig. 1. (a) Mean residence time per level, (b)(c) attack efficiency (total number of successors in all stripes / invested bandwidth), (d) absolute potential damage of our improved streaming system in comparison to [SWS07]

ing the stream and after obtaining one or several children for its agents, since it has spent nearly no effort in terms of invested bandwidth. The attack efficiency quickly decreases when the attacking nodes stay longer in the system, in order to reach more important positions in the streaming topology. This observation gives further indication of the effectiveness of our topology building principles.

In order to compare our system to the approach mentioned in [SWS07], we simulated both systems in the presence of sleeper attackers. The topology knowledge of nodes in our system is restricted to the respective fathers in all stripes, whereas in [SWS07] peers also know their grandfathers.

Exploiting this knowledge, instead of disrupting the service by switching off itself, an agent in [SWS07] could kill all its grandfathers and in our improved system an agent could only kill all of its father nodes. Figure 1(c) shows the potential damage in our system in comparison to the potential damage caused by the attacker in [SWS07]. As can be seen, the appliance of the building blocks described in 3 lowers the potential attack damage between 20% and 30% in comparison to [SWS07]. One might expect that the improvement is higher, but it has to be taken into account that many attackers have common grandfathers. As first experiments show, half the number of grandfather attackers reach the

same total damage as the original number of father attackers. A comprehensive study of these effects will be future work.

7 Conclusions

Based on a study of the possibilities of internal attackers, we identified key design decisions for the distributed repair and construction mechanisms of multiple-tree based P2P live streaming systems, including the new concept of peer testaments. Following these properties, we adapted an existing streaming system to make it more manipulation resistant.

Our experiments confirm, that the proposed mechanisms prevent active and passive sleeper attackers from rapidly ascending in the topology. More importantly, our simulation results also show that, generally, long-term internal attacks are extremely inefficient for the attacker, since its agents are forced to serve the streaming systems for a long time before reaching interesting positions.

In future, it will be of general importance, to develop a comprehensive formal model of internal attackers, by which the multiple existing varieties can be unified. Based on such an analytic foundation, the presented construction mechanisms can be further analyzed and improved.

Acknowledgements

We would like to thank Stephan Beyer and Michael Braun for their valuable help in implementing the simulation study.

References

[CDK+03] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *SOSP '03*, New York, NY, USA, 2003. ACM.

[Dou02] John R. Douceur. The sybil attack. In *IPTPS '01*, London, UK, 2002.

[LN06] J. Liang and K. Nahrstedt. Dagstream: locality aware and failure resilient peer-to-peer streaming. volume 6071. SPIE, 2006.

[RES01] V. Roca and A. El-Sayed. A host-based multicast (hbm) solution for group communications. In *ICN '01*. Springer-Verlag, 2001.

[RSS07] M. Rossberg, G. Schaefer, and T. Strufe. Using recurring costs for reputation management in peer-to-peer streaming systems. *SecureComm*, 2007.

[SM04] K. Sripanidkulchai and B. Maggs. An analysis of live streaming workloads on the internet. In *in Proc. of ACM IMC*, pages 41–54. ACM Press, 2004.

[SWS07] T. Strufe, J. Wildhagen, and G. Schäfer. Network-Efficient and Stable Overlay-Streaming Topologies (German: Netzwerkeffizienz und Stabilität von Overlay-Streaming-Topologien). In *KIVS*, 2007.

[VF06] J. Venkataraman and P. Francis. Chunkyspread: Multi-tree unstructured peer-to-peer multicast, 2006.

[WXL07] Feng Wang, Yongqiang Xiong, and Jiangchuan Liu. mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast. *ICDCS*, 2007.