Peer-to-Peer Netw Appl
DOI 10.1007/s12083-010-0070-6

REVIEW

Security and privacy issues in P2P streaming systems: A survey

Gabriela Gheorghe · Renato Lo Cigno ·
Alberto Montresor

Received: 13 August 2009 / Accepted: 18 March 2010

Abstract Streaming applications over Peer-To-Peer (P2P) systems have gained an enormous popularity. Success always implies increased concerns about security, protection, privacy and all the other 'side' properties that transform an experimental application into a service. Research on security for P2P streaming started to flourish, but no comprehensive security analysis over the current P2P solutions has yet been attempted. There are no best practices in system design, no (widely) accepted attack models, no measurement-based studies on security threats to P2P streaming, nor even general surveys investigating specific security aspects for these systems. This paper addresses this last aspect. Starting from existing analyses and security models in the related literature, we give an overview on security and privacy considerations for P2P streaming systems. Our analysis emphasizes two major facts: (i) the Byzantine–Altruistic–Rational (BAR) model offers stronger security guarantees compared to other approaches, at the cost of higher complexity and overhead; and (ii) the general perception (not necessarily the truth, but a commonplace belief) that it is necessary to sacrifice accuracy or performance in order to tolerate faults or misbehaviors, is not always true.

G. Gheorghe · R. Lo Cigno (✉) · A. Montresor
Dipartimento di Ingegneria e Scienza dell'Informazione (DISI),
Università degli Studi di Trento, Via Sommarive 14,
Povo, 38100, Trento, Italy
e-mail: renato.locigno@disi.unitn.it

G. Gheorghe
e-mail: gabriela.gheorghe@disi.unitn.it

A. Montresor
e-mail: alberto.montresor@disi.unitn.it

Keywords P2P streaming · IPTV · Security · Privacy

1 Overview

Peer-to-peer systems have gained more and more momentum over the last years as a means to access multimedia contents, albeit initially in form of file downloads. The evolution to streaming and multicast (e.g., TV) was just a consequence. Their power to accommodate large amounts of users, together with their resilience to churn, reliability, and low cost are some of the reasons why they are preferred over dedicated servers or content distribution networks solutions. In spite of these advantages, or maybe because of them, some P2P features make these systems more difficult to defend against some classes of attacks.

Security-wise, P2P streaming systems are more challenging than other P2P applications because they are more vulnerable to QoS fluctuations. Live streaming protocols, and TV in particular, are most sensitive to delay and delay jitter: it is enough for a host to be prevented from receiving some packets in time, and the user may grow dissatisfied with the quality of the delivery and leave the system altogether. If some other peers are connected to that machine, they will be damaged as well. From the watcher's viewpoint, even slight quality fluctuations, or choppiness, cause the viewing experience to loose appeal and the user to drop the service (or switch channels if others offer better quality). Worse, the quality of the user experience is unrecoverable: if some packets are lost during live broadcast, they are lost for good because recovering them afterwards brings no utility to the user.

Published online: 23 April 2010

Apart from their time-sensitive nature and bandwidth dependency, P2P streaming are susceptible to manipulation and threats at the transport and network layers. Clever attacks can compromise selectively the guarantees that a streaming session should provide, making some channels unusable, or unavailable in particular locations. Both events can be classified as targeted censorship violating the freedom of speech and expression. Analyzing the threat models in all these cases gives relevant indicators over possible risks and vulnerabilities in the transmission.

In what follows, we provide a brief security analysis of P2P live streaming, and provide a classification of both attack points and solutions to common vulnerabilities. We provide general considerations and features that novel P2P streaming proposals should consider in order to minimize the chances of attack.

The remaining of this paper is organized as follows. Section 2 motivates this survey by introducing a few attacks examples, either directly to P2P streaming applications or strongly related to them. Section 3 discusses the threats and security models for P2P streaming applications, with specific attention to P2P-TV systems. Section 4 describes the easier, and hence readily realized, attacks in P2P streaming systems. The study continues with discussing security practices (Section 5) particularly for the tree and mesh overlays (these latter combined with data-driven dissemination). We conclude discussing the trade-offs of this type of mechanisms, open issues and future work.

2 Examples

Finding real-world examples of security attacks to P2P streaming systems is (unfortunately?) not easy, because these systems are young, and most of all because the organizations managing them are not so keen in releasing information about attacks to their systems. In the following we sketch two examples based on real life events that are clearly related to P2P streaming and help us introduce the reason why we consider important addressing security issues in P2P streaming systems and do it *before* large scale attacks make the headlines of non-technical literature.

2.1 Example 1: the reason for polluting

Albeit rarely admitted or clearly proved, it is commonly accepted that in file-sharing applications content pollution, i.e., intentionally change parts of the file to make it useless or of bad quality, is a day-by-day routine. In [11] and [36] it is practically given for granted that a P2P streaming system can undergo pollution attacks.

It is often reported that pollution attacks in file sharing are due to the fact that the system is devoted to illegally exchanging copyrighted material, and it is the copyright owner who pollutes the system as a last means to defend its rights when any legal action failed.

If this were the situation, then one may think that in a system distributing legal content like standard public TV there is no reason to consider pollution. This position is however rather naïve. Making some specific content unavailable can be a goal for many actors. For instance changing the advertisements on very popular events can lead to very remunerative commercial frauds. On a larger scale, selectively changing (or simply removing) parts of some content may lead to public opinion manipulation that, if done by a government or similar body can be called *censorship*, but if done by private (criminal) organizations raises even more frightening scenarios.

2.2 Example 2: Skype outage

In recent years, the problem of facilitating signaling in VoIP (Voice over IP) networks through a P2P network has been subject of intense activity in both research [33] and standardization [17]. While clearly the problem domain is different from video streaming, they share similar security concerns, among which time-sensitiveness is the most important.

One of the main attacks that can be played against P2P VoIP systems is *denial of service* against the *availability* of the signaling system: the attacker may try to block the ability for a caller to identify the current location for the designated callee. Furthermore, given that the caller expects to retrieve this information in reasonable time in order to start the call, it may be sufficient for the attacker to severely delay the transmission to the location of the callee.

To achieve this goal, one of the easiest forms of attack is to try to perturb the routing substrate of the P2P system, normally based on distributed hash tables like Chord [10] and Pastry [26]. Possible attacks in these cases include Eclipse, Sybil and neighbor selection attacks. These problems, casted in the video streaming domain, will be discussed in Section 4.

An example of the kind of problems that users can expect from VoIP systems, which can be mirrored in the video streaming domain, is exemplified by the Skype two-day outage which occurred on August 2007 [1]. While Skype has denied that this specific event has been caused by malicious activity, blaming instead the “Microsoft Patch Tuesday” (with a large number of machines rebooting at the same time). This is an example of what could happen when nodes in a P2P streaming service (voice is being streamed here) loose *autonomy* (see Section 3.2). The loss of autonomy, which leads to a *dependability* problem in this case, is due to the dominance of an operating system in correlating nodes. In general, nevertheless, the same problem can be due to any other reason.

3 Security considerations for P2P streaming

Security design for a P2P system builds on the relevant aspects of that system: first, the actors posing the attack and the assets to be protected are the starting and ending points of any attack. Studying sources and targets of attacks is usually done by means of the *threat model* (Section 3.1); second, any attack intends to subvert or damage an existing scheme, often for the benefit of the attacker; the result is that the system will no longer function as designed, hence it will not be able to fulfill its *goals*. Knowing what these goals are helps identifying the ultimate result of a security breach. Sections 3.2 and 3.3 overview the possible non-functional goals that a streaming system aim at. Finally, Section 3.4 shows the possible mechanisms to protect the system assets for achieving the goals presented before.

3.1 Threat model

In any security analysis, it is important to consider all possible generators of attacks (*active* elements) against possible targets of attacks (*passive*). In the first category we find P2P nodes, supernodes and application code, while in the second we include the protocol, the overlay, and the data being transferred. Although the streaming source can also be a possible target, the usual assumption is that the source is trusted, since we have not found any studies on source-level attacks. The application code can be seen as both active (when causes data leakages, or jeopardize data privacy) element, or passive element of attack (when can be directly manipulated for protocol subversion). An overview of the threat sources and targets in P2P streaming applications is given in Tables 1 and 2 and is detailed hereafter.

There are three major elements likely to turn into sources of attacks in P2P streaming:

Peer nodes Malicious or malfunctioning nodes can always alter the protocol behavior. For instance, they may not reply to requests, or may reply generating wrong messages. This can result into biasing the neighbor selection process of another node, thus into network partitioning or even censorship. Censorship has deep consequences: besides the standard legal aspects, a smaller number of users in the overlay implies a poorer quality of the diffusion [22]. From the point of view of QoS, peers can also do delayed forwarding and hence jeopardize once more live streaming and TV systems.

Table 1 Common sources of vulnerabilities in P2P streaming

Active aspect	Influences what	Results into
Peer nodes	P2P protocol QoS	partitioning, censorship delays, isolation
Supernodes	P2P protocol QoS	partitioning, censorship delays, isolation
Application code	P2P protocol data privacy	censorship data leaks

The active aspect in the first column indicates the class of entities that generate attacks; the second column refers to the targeted system feature, while the third column shows a possible result of the vulnerability being exploited

Supernodes Supernodes do not always exist in P2P applications, but it is envisioned that they can greatly benefit applications requiring large bandwidth and low, constant delays. Supernodes bring similar vulnerabilities to streaming systems as common peer nodes; however, the emphasis is on their higher responsibility in data diffusion: e.g., if superpeers do not behave fairly and honestly with all peers, they can bias the service toward preferred users. As a consequence, partitioning and censorship are more stringent at the supernode level. Supernodes become even more critical as some projects explore the possibility that they are controlled by ISPs in an effort to make P2P overlays and IP networks cooperate [20].

Table 2 Common attack targets in P2P streaming

Passive aspect	Influenced by	Results into
Application code	Code provider	censorship data leaks
P2P protocol	peers superpeers application code	censor, partition, pollution partitition, DoS data leaks
Overlay routing	data privacy QoS overlay routing	data leaks delays, DoS partitioning, censorship
Distributed data	data integrity	partitioning

The aspect in the first column indicates the vulnerability, the second column refers to possible sources of attacks, while the third column shows a possible result of the vulnerability being exploited

Application code Wallach [34] notices that the P2P code runs with numerous privileges on peer machines: it normally uses the network connection and the local hard drive. When unrestricted, local access and external communication may lead to information leaks or malicious code installed on the local machine that could alter the overall P2P protocol. The remedy is twofold: sandboxing the P2P application to use just an isolated location on the local drive, and denying operations that are not coherent with the purposes of the P2P application. The application code poses a particular threat to users' *privacy*, because embedded malware could leak sensitive information to non-authorized recipients.

The passive sources of vulnerability that are usually targets of attacks are:

Overlay routing and maintenance Overlay management messages among peers aims at reliability and quality. Secure routing deals with both maintaining secure routing tables, and securely transmitting messages [34]. The data in transit can be sniffed and if the channel is not secure, it can even be leaked or modified. The dispatching of tampered data to fair peers depends on the security of the overlay and neighborhood tables; not only routing can undergo malicious delaying, but also partitioning (sending tainted data to the same peers) and/or censorship (not sending anything to a group of peers).

The P2P protocol One way or another, the attackers in a P2P scenario always try to manipulate the protocol to their own advantage, or to the disadvantage of other peers. The P2P application protocol is at a higher level than the overlay routing mechanism, and manipulates streamed data by correlating a number of aspects: membership mechanism, data scheduling and transmission, identity management, overlay mechanisms, reputation, etc.

Distributed data Data integrity is essential in streaming and TV systems, because the purpose of the application is liveness. If a TV-channel is re-distributed on the P2P system but part of the news/programs are altered with some users treated differently from others, this can lead to partitioning, loss of users, and censorship.

3.2 System-level security goals in P2P streaming

If the threat model identifies the sources of potential jeopardy to the system, the *security model and goals* identify the aspects of the system that are jeopardized by the threat. In Table 3, we split these aspects into system operation, introduced here, and content management, discussed in Section 3.3.

Table 3 Desirable security and privacy features for P2P streaming systems

Category	System feature
System operation	Reliability, Availability, Dependability, Node Autonomy, Access Control
Content management	Authenticity, Integrity, Non-repudiation, Confidentiality, Anonymity

For what system operation is concerned, we identify the properties listed in the first row of Table 3 and discussed in the sequel as those to be granted to streaming systems in the face of threats and attacks. The system security, in the context of streaming system operation, can be identified with the capability of the system not to *fail*. We note that the term *fail* assumes a different flavor in streaming systems, specially for live events, than in other traditional P2P systems. Indeed, considering a file-sharing application, failure can be identified with the inability to download a file, or at most with the inability to do that in a given time: it does not matter whether the system operates continuously or in bursts, or if the file is downloaded at a regular pace or all of a sudden right before the deadline. To make another example, in telephony applications it is not a security requirement that all users can talk at the same time (the probability of such an event is considered negligible). In a streaming system, instead, the inability to connect to a stream by *any* user is a failure, even more so if this happens for a very popular, and hence important stream, which might lead to users discrimination.

We use the terminology for systems failures as defined by ITU-T in the E.800 series[1] and by IFIP WG 10.4.[2]

Reliability The up-time of the system in steady state is the reliability of the system, and as such it is normally modeled by the mean time between failures (MTBF). Reliability can be a property of a single device or subsystem, a global property of the entire system or, as more suitable for our purposes, it can refer to the vision of the system conditioned to one specific peer or a subset of peers. A single failure, even if recovered, implies loss of reliability. Let us characterize reliability as

[1] http://www.itu.int/rec/T-REC-E.800/en

[2] http://www.dependability.org/wg10.4/

$\rho = 1 - \frac{1}{\text{MTBF}}$, where MTBF is the number of consecutive requests from a peer before one fails (hence MTBF ≥ 1). Reliability is a desirable feature for security, but asking high reliability to a highly volatile system, like a P2P overlay, which is designed for resilience rather than resistance is not appropriate, thus the main goal of reaching a certain level ρ' of reliability is ensuring the system high *Availability*.

Availability It is the ability of the system to be up and running. A system can be unreliable, yet highly available, simply because recovery from failures is faster than the user/application of the system can detect. In P2P streaming, for instance, churn can be a source of unreliability, since peers leaving implies that from the point of view of some other peers, a portion of the system (or a given request) has failed. However, topology reconfiguration can be fast enough to avoid the loss of any information at the application, so that the system remains available even from the perspective of peers that are affected by churn. In general terms, we can say that a secure P2P streaming system must be available with high probability at any time, contrary to other (not all) P2P applications, most notably file sharing, where the system can be unavailable for relatively long periods, but still operate securely in that it yields its services.

Dependability Even if highly available, a system may still suffer from correlated failures that make it non-dependable. Dependability is a subtler property of the system: it reflects the ability of a system to work and provide services in critical moments. An example will clarify the point. Cellular telephone systems are in general reliable and available, however they are not dependable with respect to emergencies and civil protection: during accidents the cells covering the area of the accident become congested because people call with higher rate than normal and resources are locally insufficient; during natural disasters, besides the above phenomenon, normally the electricity fails, and the base stations do not have adequate power backup. In the context of P2P streaming and TV applications, the system may turn to be non-dependable because simple attacks can ruin specific event streaming (e.g., popular broadcasts) which causes a higher-than-average amount of traffic; in these cases, simple traffic-volume based attacks can jeopardize the most useful (or prized) events.[3]

Dependability is a security feature more critical for multicasting and broadcasting systems than for other systems because of the correlation between the value of the events and the number of peer/people wanting to receive them. Moreover, if the streamed event relates to critical public news, then the failure of the system represents not only a lack of security, but also a public/social safety problem.

Node autonomy This is a system security goal that is somewhat specific to P2P systems. Each node is peer with all the others and its autonomous functioning should be guaranteed at all times: at any point during service, each node should be empowered to perform the actions that it is specified to perform at that step, without the need of external intervention. This does not mean that the node is isolated, nor that it cannot interact with other nodes or with external repositories, like in the ALTO IETF architecture [27], but that the node is not depending on this information for its operation. Dependencies on external intervention expose the node to trivial DoS attacks (when the information is not available, the node cannot work), and many other security threats. For instance, node autonomy is a requirement to prevent censorship attacks, and as discussed in the second example (Section 2.2), the ability of decorrelating reboots or similar actions is fundamental to avoid massive failures that lead to information loss.

Access control Access control is fundamental to avoid frauds in commercial services, and fraud avoidance is a security goal. Access control can be a conflicting requirement with Node Autonomy. On the one hand, to the best of our knowledge, there are no known methods for distributed authentication, so that, for this function, the node cannot be autonomous. On the other hand, it can be argued that a commercial system requires a form of centralized control (by the service provider) and is provided in exchange of some form of payment (direct or not). Thus in this case there is a commercial agreement between the two entities and any form, for instance, of denied access can be tracked and is not a DoS.

Access control is also a powerful means to reduce the possibility of security attacks coming from inside the system, because it prevents identity misrepresentation as well as, to some extent, collusion and multiple identities. The real challenge is providing access control while preserving the users privacy, i.e., implementing a

[3] A volume-threat is a subtle form of DoS attack: the attacker does not need attack directly any peer or the source, it just needs to inject in the network, which is already loaded because of the very popular live event streamed, enough dummy traffic to cause a packet loss rate that the streaming application cannot cope with.

system that either guarantees against information leakage (e.g., what TV channel is downloaded), or enables pseudonym-based authentication [3, 4].

3.3 Data and content security goals in P2P streaming

Considering now content management, there are some specific properties of P2P streaming that are of particular security interest:

Authenticity and integrity The data transmitted must be guaranteed and not tampered with, and it must be guaranteed that it was emitted by the intended transmission entity.

Non-repudiation Refers to the situation when the nodes that received a certain piece of data cannot deny that they received it. Non-repudiation is of interest only for video on demand applications, while for TV-like (broadcasting) it may be a minor feature.

Confidentiality The content that is transmitted during the streaming process can only be used or retransmitted to other nodes involved in the protocol. This property interlaces with access control. In fact an access control system that prevents unauthorized participation to a streaming, but is not supported by a content management system that can prevent recording and later replication of the content becomes useless. Recent studies on commercial TV streaming solutions have shown that they do not perform encryption [7], which makes the protocol lose not only confidentiality but also authenticity and integrity.

Anonymity This is one of the most controversial properties, since in many contexts the capability of a user to remain anonymous is associated to potentially unlawful activities. However, specifically in TV systems, the right of a user to watch a program without disclosing his identity is key to privacy protection and should be guaranteed by broadcasting systems. This property should be guaranteed also by P2P streaming systems, not only in face of external observers, but also with respect to the other users of the same system, and the broadcaster too.

Haridasan and van Renesse argue that not all applications need anonymity and confidentiality, but the features that matter most in frequent cases, are authenticity, integrity and non-repudiation [15]. Still, we have seen that anonymity becomes a key issue of privacy protection in TV systems. Non-repudiation, in the same systems, may be of secondary concern, unless a node can build claims on the fact that some information has not been delivered. Similarly to Section 3.1, these are clear differences of security requirements and other P2P applications.

3.4 Protection mechanisms

As we have discussed, in P2P streaming there are two important values to be protected: i) the data exchanged between peers, and ii) the hardware and software resources that each user somehow ‘lends’ to the P2P system.

In streaming systems the data being shared has a limited validity in time: after the target playout time the data turns stale. This adds a new dimension to the problem of data protection: delay makes data useless. As a consequence, *bandwidth* becomes an asset that can be attacked to make data useless. As P2P systems are decentralized, it is usually easy for malicious peers to flood the system with junk and fake data in such a way that they would exhaust the bandwidth of the system [8].

Access control is a prevention mechanism that limits the reach of unwanted entities (peers) to the data being exchanged. Mapping and enforcing the connection between identities and access rights, access control strongly requires mechanisms for identity and reputation management. Some applications (e.g., public TV) require no access control for service provision, but others may be limited to groups of authorized users: membership is controlled, and the system should provide means to protect membership in face of attacks, both for breaking the control and for denying service to authorized members.

Auditing Auditing is a detective means by which violations of predefined courses of actions can be identified. Unlike access control, auditing is an ‘after the fact’ measure and the outcome of its analysis influences future course of actions. Auditing requires the existence of logs with recordings of certain activity, the mechanism that is periodically triggered to write these logs, and an auditor—the entity verifying the logs. As far as the checking mechanism is involved, auditing can be continuous—at certain time intervals or on all records—or probabilistic—at random moments of time or on random recordings.

In P2P systems, audit can function as a means to check whether a peer node functions according to a predefined contract or protocol. The idea of distributed audit in the sense that nodes trade local storage with storage on other nodes, is hinted in [34]. Of course, in order to perform it, the auditing method must be secured; this involves making sure that any nodes cannot influence what is being written in the logs, nor hide the logs themselves. Full access to query these logs must be entrusted to the requesting entity; moreover, the mechanism evaluating the events logged in the file must not misinterpret or ignore anything that was recorded. A simple way to ensure that most of these conditions are satisfied, is to impose a reward/punishment/incentive mechanism that makes the entities involved in the audit process cheat as little as possible.

4 Common attacks in P2P streaming systems

The most serious attacks in P2P systems comes from the inside of the system. This happens because only an internal node runs the protocols used between hosts, and can thus exploit them. It is common practice to restrict the set of actions that unknown nodes can perform. Once a node is allowed into the system, however, it is assumed to be honest hence it gets all privileges. With the problems presented in Section 2, the bad nodes become bad on the fly. Therefore, the security of P2P application should look to protect internal nodes from other (malicious) internal nodes (Table 4).

In what follows we will focus on some possible situations of vulnerability and describe the favorable conditions in which they take place. Each of the following attacks can exacerbated by *collusion*: one malicious entity compromises a (potentially large) collection of nodes to conduct correlated attacks onto the whole system. This scenario breaks the node autonomy requirement stated in Section 3.2. As expected, this is the most dangerous situation since it may be extremely difficult to track down the attacker if nodes function correctly at each step or on short-term, while overall misbehaving or deviating the protocol on the long run.

Forgery and repudiation attacks Forgery attacks break the condition of confidentiality and integrity of data mentioned in the previous section as a requirement of P2P streaming systems. Haridasan and van Renesse call *forgery* any fabricated or tampered data streamed into the system [14]. Repudiation attacks are attempts to deny having received streaming content or to acknowledge but with false information. Most cryptographic techniques as message signatures and public key infrastructures can easily solve the vulnerability, but suffers from the disadvantage that the performance cost of signatures or keys is high.

Table 4 Overview of attacks in P2P streaming systems

Attack	Target	Attribute
Forgery	data	confidentiality, integrity
Pollution	data	confidentiality, integrity
Eclipse	overlay, protocol	autonomy
Neighbor	protocol	autonomy
Sybil	protocol	authentication
DoS	peers	availability
Omission	peers, data	dependability

Attackers can collude in pollution, membership, neighbor selection, Sybil and DoS attacks. The source of attacks is usually any peer node. In some cases—pollution, forgery, neighbor, omission—superpeers can do more damage to the system than average peers

Pollution attacks in P2P streaming occur when the attacker mixes or substitutes junk pieces of data into the P2P distributed stream. In this way, the quality of the transmission decreases considerably: polluted chunks which arrive at fair peers degrade the stream quality and can change its meaning; and these peers will forward the *junk* to other peers and the whole effect will exponentially span over the network. Proof that the effects of this type of attack can be devastating in a streaming scenario are given by Dhungel et al., along with proposing four possible defenses: blacklisting, traffic encryption, hash verification and chunk signing [11].

Membership and Eclipse attacks With this type of attacks, the membership protocol or the way nodes are admitted into the overlay are compromised. A special type of membership attack is the Eclipse attack, where, as noticed in [32], an attacker which controls a portion of the overlay neighbor scheme, *eclipses* fair nodes by dropping or re-routing any messages meant for those nodes. In other words, in Eclipse attacks, the attacker can gain some control over the routing mechanisms in the P2P system.

Unstructured overlays are more susceptible to this type of attacks than the structured overlays; the latter do impose some constraints over the neighbors of one node, while the former do not. For this reason, the unstructured overlays use floods of random walks to gain knowledge of the network topology; the more they use these mechanisms, the higher the probability that an attacker will control more nodes in the system. One possible solution described in [32] is to use a mechanism that bounds the in-degree and out-degree of the nodes in the P2P overlay. In this way, an attacker is prevented from communicating with more nodes than those to which it normally should.

Neighbor selection attacks These attacks refer to the situations in which an attacker controls the neighbor selection mechanism of some nodes, and makes them choose it as information provider. Malicious nodes can thus infiltrate and dominate sets of neighbors. The attacker will influence the way the overlay communicates and the neighbor selection process happens, so that it can control the traffic and subvert the whole system. These attacks are referred to as *epidemic* by [29], as fair nodes will "unknowingly reference compromised peers in their neighbor set". Of course, the problem is even worse if the membership server is itself attacked in this way. One idea of solving this problem with Distributed Hash Tables is to identify the invariants in the placement of peers in the overlay, and detect attacks in the form of deviations from these invariants. A solution adapted to mesh-based systems is shown in [29].

Sybil attacks These attacks happen when the reputation mechanism established within the P2P system is compromised. Specifically, an attacker creates a large number of entities which bear the same disguised identity in order to become more powerful. Depending on how the id-s of nodes and reputation constraints are generated, the reputation system may be more or less vulnerable to such attacks. The idea is that once disguised, the attacker profits from the trust that is given to the real entity it impersonates. Guarding against such attack may involve a trusted third entity which certifies that a name or a reputation id is attached to the exact entity it is supposed to carry it. Therefore, certified node identifiers is one of the most straightforward techniques to repel masquerading. In addition to this method, auditing is another way to prevent the Sybil attack. An interesting solution employing auditing is provided in [32], where a node periodically challenges one of its neighbors to provide it with a list of that node's inbound contacts; if that list appears unfair or tampered with, then the requester node can act upon this discovery.

DoS attacks Denial of Service can take many forms, from system partitioning to sending excessive amounts of requests or duplicate packets intended for their peers. The ability to bring a contribution to the streaming session is thus compromised, because a fair node would be flooded with useless messages or too many requests for it to handle. In this way, the resources of the system are exhausted with a relatively small effort on the attacker side. When the resource on which the attack focuses is bandwidth, the attack has been also termed as *request spreading attack* [8]. These problems were previously studied in the case of distributed systems as well as P2P streaming scenarios and there are several approaches in counteracting this type of attacks [8, 9, 35].

Omission attacks Are at the other extreme than DoS attacks, implying that all the packet of data or just a part of it is not sent further according to the protocol specification. Again, just like for the DoS attacks, this behavior can compromise the whole P2P system even if a small number of peers collude. As noted by [15], the problem with this attack is that the guilt of a node cannot be proved easily.

5 Security practices

In this section we provide a close-up on the existing security solutions in P2P streaming. We expose and discuss the vulnerabilities of each approach and then derive a few patterns and conclusions that would help in protecting against attacks in P2P streaming systems.

As shown in Fig. 1, there are two building blocks of P2P systems to be considered: overlay topology and data dissemination mechanisms. The topology of the overlay defines how to connect each node in the network with the right neighbors; in other words, in a situation in which nodes are constantly joining and leaving the system, to find a solution in which each node sees as its neighbors only the nodes it is most interested (and is fair) to communicate with. The criteria to choose neighbor range from locality to certain QoS values.

The topology of the overlay is in tight connection with the application: the application domain determines the topology of the network, while in its turn the overlay topology influences runtime application aspects that can be either functional or non-functional: searching, routing, performance, efficiency, robustness [5, 6, 16]. In complex applications, where the topology changes dynamically, the mechanisms involved in the construction of the overlay have increased importance because they are invoked continuously; consequently, keeping these mechanisms protected against attacks becomes essential in order to maintain their compliance with the protocol schemes.

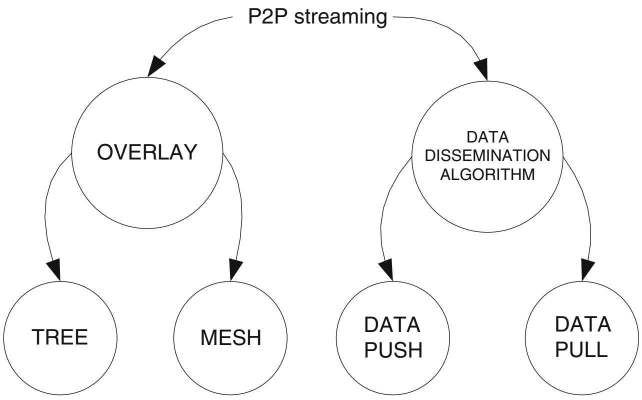

Fig. 1 Important structural aspects in securing P2P streaming

According to several classification studies [22, 24, 28, 38] that there are two typical overlay topologies in P2P streaming applications:

1. Tree-overlays: in which the overlay is usually built in the shape of a tree. This means that the way in which overlay nodes send and receive messages is structured and embedded in the overlay topology: The source is the root of the tree and leaf nodes receive but not redistribute the data. Other structured topologies, like multi-trees and hypercube exist;
2. Mesh-overlays: where the overlay does not have a specific structure but it is a generic mesh. That is, every peer has several neighbors, but without a clear parent-child relationship or any predefined topology. The media is distributed among different peers and then each of them transmits the media further.

Apart from the overlay construction, the other defining aspect of P2P systems is the *data dissemination mechanism* among peers. That is, while the overlay deals with connecting a node with the right neighbors, the data dissemination algorithm is concerned with how to select neighbors to actually exchange information with. There are two basic ways of disseminating data in P2P streaming systems [22, 31]:

1. The *push*, or source-driven approach means that a peer transmits a chunk to its neighbors, assuming they do not have it yet; the directions in which the data is sent are determined by the parent-child relationship among nodes, be it a tree or mesh overlays. It is easy to see this way of performing data dissemination is prone to redundant pushes and thus to DoS attacks (e.g., flooding neighbors with data they already have), to neighbor selection and omission attacks (bias in where to push data);
2. The *pull*, or receiver-driven approach is an alternative to the previous scheme, by which a peer uses buffer maps to create pull schedule with the peers it decides to communicate with. A peer requests the information it is missing. This approach is more robust than the previous, but vulnerable to collusion: peers that already have data may not advertise it to others;

The *data-driven* approach is in practice a pull mechanism. Epidemic algorithms (and gossiping ones in particular) are examples of this approach. Gossip in P2P is a data dissemination mechanism that does not rely on the overlay structure but autonomously manage its own distribution patterns. It is also useful in data aggregation and resource allocation [19]. The reason for its popularity is that gossiping mechanisms are simple and more robust than others. Security-wise, we believe they are interesting to study because they are more general than the push and pull mechanisms. The biasing vulnerabilities suffered by the other approaches are easily solved with gossip, since it is not easily predictable in which way data flows. In addition, as previously noted [31], gossip-based mechanisms are less sensitive to peer dynamics, thus to churn.

For the reasons above, in what follows we will analyze the two overlay approaches in conjunction with gossiping protocols from a security standpoint. We will bring into light what are the vulnerabilities and strengths induced to the systems that adopt these approaches.

5.1 Tree-based approaches

Generally, streaming in tree-based overlays imposes that the source of the media is the root of the tree, and that the rest of the peers are children of the source and children/parents among themselves. The path that the data must follow in this case is fixed: first from the source to the first-order parents, then from those to their children, and so on. A visible functional problem that occurs with this kind of overlay structure is simple: the efficiency of the hierarchy is overcome by the large imbalance between parent nodes and leaf nodes (parents forward data while leaves do not, so everybody wants to be a leaf). Historically, the solution to this issue took the form of *multi-tree* overlays, as [28, 29] notice, in other words: more trees, more leaves. This approach leads to distributing the data in multiple distinct trees.

There are other problems related to the topology of this overlay [22, 38], and they are summarized in Table 5. Since in tree overlays each node receives data from only one source node, bandwidth fluctuations can be highly damaging, and paths that are closer to the

Table 5 Common fairness and security issues in tree-based P2P streaming systems

Problem/attacks	Envisaged solutions
Imbalance root vs. leafs	Using multi-trees, gossip
Bandwidth fluctuation, bottleneck	
Protocol deviations on parent node	
Identifying malicious nodes	Monitor, acknowledgment
DoS, omission	
Membership attacks	
Forgery, repudiation	Signatures
Sybil attacks	Not yet solved

root are more likely to turn into bottlenecks. Security-wise, minor protocol deviations of single nodes can affect easily entire subtrees. Additionally, when nodes closer to the root leave the system (e.g., they crash or are attacked), they leave unserviced a large percentage of the nodes.

Trying to solve the above problems, Zhou and Liu have combined the tree overlay with gossip data dissemination so that the two approaches would compensate each other's faults [38]. Because the tree model is brittle but yet time-efficient, it is used as a second option: by default all data is transmitted by gossiping, and if a node does not receive anything for a certain period of time, the tree overlay will be used to obtain the data from its parent. Security-wise, because the protection level for a composite system is the protection level of its weakest link, this solution is prone to all vulnerabilities of the tree overlay.

Another solution adopted in tree-based overlays is presented by Shetty et al. in [30]. In tree-shaped overlays, the streaming quality depends on the cooperation of the non-leaf nodes (namely the nodes in the overlay tree that are neither leaves nor the source). The possible attacks that are considered are thus DoS, omission, forgery and repudiation attacks. Shetty et al. identify that one of the problems with the current security solutions in P2P streaming is that they cannot identify the malicious nodes themselves, just the fact that there are malicious nodes. This is because in overlay multicast streaming, if a fair peer receives tampered data, it cannot determine if its parent is malicious (since its parent might have taken that data from some other peer).

A *signed acknowledgment* together with a *random monitoring* scheme were shown to be a solution to detect the exact attacker peers. The former mechanism is used by peers to prove their fairness, while the latter helps trusted peers to monitor in a random fashion some of their peers suspected to malfunction. The problem with this solution design is that it relies on *one single session trust manager*, which imposes a scalability issue and a single point of failure. The trust manager decides whether a peer is malicious or not, by receiving 'complaints' from peers and employing a localization scheme. If it cannot detect the exact location of the omission or forgery attack, the trust manager will decrease the trust value of both peers (the reporter and the reported). Otherwise, the child and the tree that inherit from the reported node are moved to another peer-tree.

On the downside, this solution does not handle collusion attacks in the form of Sybil attacks: if a malicious node assigns itself several identities, then the only way to prevent it from gaining control is by assigning strong identities from a central identity manager, or conversely, to implement a punishment scheme, where a malicious node can be evicted, provided bad service or punished in money. Moreover, having the trust manager as one fixed peer throughout all sessions is a single point of failure, therefore passing this responsibility to different nodes with high levels of trust for each session, should be a straightforward improvement.

A common solution in tree-based approaches is given by SecureStream [15]. SecureStream is able to repel several types of attacks because of its multiple intrinsic mechanisms that help in eliminating most vulnerabilities. For example, in order to protect itself against membership attacks, SecureStream uses the *Fireflies* protocol in which members monitor each other in case of failure, by pinging each other. The pinging protocol is based on a gossiping protocol, where each node is assigned certain other nodes to monitor, and there is a limitation on the number of neighbors that a certain node can accuse of failures (this comes to stop malicious nodes from accusing too many fair ones). Each peer has a predefined set of neighbors.

To guarantee the integrity of the data being streamed over the network, SecureStream avoids signing groups of packets with asymmetric keys, but computes content hashes that are signed with the sender's private key. In order to minimize the number of malicious neighbors, this solution takes a smart approach: in each round, the source of the transmission notifies its neighbors that it has available information. This information is valid for an "availability window". Each neighbor, in its turn, requests from the source the information that it misses (this is the "interest window"), but trying not to overload the source. Whenever they obtain new information, peers send notifications to their neighbors. Overall, this method of dissemination is resilient to attacks, especially omission attacks: if a peer does not reply with the promised information, then another one is contacted. It should be noted here that the nodes located close to the source do not necessarily receive packets faster.

Moreover, there is a limit in the number of requests that arrive at a peer (this repels the possibility of node flooding). In order to eliminate free riders, an auditing mechanism ensures that all nodes contribute to the protocol at least as specified by a minimum amount. Auditing is distributed: local auditors are periodically elected to evaluate the contribution of each of their neighbors. Punishing nodes that behave maliciously, as well as employing the pull-approach that disables attackers to gain control deterministically over sets of nodes, are the salient features that make SecureStream tolerant to Byzantine attacks. In comparison to the BAR-Gossip approach described in Section 5.3, it can be noted that the source does not need to know all the members of the protocol: here the membership is dynamic, so scalability is not bounded.

5.2 Mesh-based approaches

Mesh-shaped overlays are less structured in comparison to the tree solutions. A membership server may keep track of the existing nodes in the system if required, and there is no fixed flow that data must follow. Recent works see these meshes as unidirectional, in the sense that nodes have separated inbound and outbound links. The number of neighbors that a node can accept is limited by resources or by the protocol.

Empirically, a comparison between multi-tree and mesh-based overlays in streaming scenarios is given in [24, 28] and the conclusion is that mesh approaches are more robust. The study shows that overlays that are mesh-shaped bear better performance when the size of the network is large, the streaming rates are high, and the nodes have high bandwidth and low round-trip times. On the downside, they may introduce a large number of duplicate packets in the network. Multi-trees, in comparison, are more time-efficient in heterogeneous networks, but on large scales they perform worse than meshes. Some issues of the mesh-based overlay are shown in Table 6.

Two classical examples in mesh-based overlay solutions are Prime [23] and CoolStreaming [37]. In CoolStreaming, the approach is data-driven: the data availability drives further propagation; gossip communication is used to disseminate network membership and content availability. Building on CoolStreaming, which does not form a typical mesh but several trees onto an initial mesh, Prime is historically one of the first mesh streaming systems. In Prime, content delivery (or swarming) has two phases: push reporting is done by parents (announcing availability of data) and pull-reporting by children (retrieving data using some packet scheduling algorithm). For advertising the new content dedicated links are in place (diffusion connections) over diffusion trees.

Table 6 Common security issues in mesh-based P2P streaming systems, and possible solutions

Problem	Envisaged solutions
Identifying malicious nodes Flooding, omission attacks Membership attacks	Monitor and audit schemes
Collusion attacks Data diffusion problems Acknowledgment / Repudiation pb.	Not yet solved

From a security point of view, neither Prime nor CoolStreaming protect themselves from effects of several types of attacks. For example, Prime assumes that peers are all fair and connect in a random fashion with one another, and that the mesh formed by peers is directed. Since there is no mechanism to check whether instead of randomness, some nodes can connect only to certain other nodes on purpose, so coalitions (and also network partitioning) can form. Apart from the simple collusion attack, the integrity of the diffusion connections is not enforced. There is no mechanism in place to make sure that one node declares its content availability to all or none or a fraction of its neighbors; there is no guarantee that the bandwidth, outgoing and ingoing degree of each node are used properly. Even more importantly, there is the issue of acknowledgement and repudiation: there is no guarantee that peers eventually receive streamed data.

5.3 Gossiping and Byzantine faults

Gossip algorithms are mostly used for content dissemination in dynamic distributed systems. They rely on what is termed as “probabilistic exchange of information” [19]: nodes use randomness in determining to/from which neighbor they would forward/retrieve data. Gossip protocols in general are robust, scalable and rapidly spread information, their only fault being that they might generate more traffic than the nodes can handle. This traffic quantity, however, is a price that gossip protocols pay for redundancy. Still, even if dangerous, they can become a very useful tool for rapid and scalable epidemic dissemination when proper attention is given to the message propagation mechanism.

Based on an analytic model [19], there are three main features of common to gossip protocols:

1. Peer selection is about how a peer A selects another peer B in order to interact with it. This choice must be done randomly in a typical gossip protocol, but it can be biased if the scheme is undergoing an attack: peer A might see that there are three other available peers B, C and D, but its choice on communicating with B might not be a random choice. For this reason, securing the gossiping protocol involves adding a mechanism to enforce that peer selection is random and cannot be tampered with;
2. Data exchanged is an application-dependent choice belonging to each of the two peers involved in the exchange. It is not necessary that peers exchange

data: they might as well exchange references to other peers in a better ‘position’ to exchange actual data (e.g., more bandwidth). From this point of view, it is essential that none of the two parties cheats. This constraint can be enforced by checking the validity of the data during the content exchange, before the communication between the two peers ends. In addition, the security of the channel established between the two peers must be enforced;

3. Data processing refers to how each peer handles the data it has received. It involves both storing the message for the next round and passing it to the application. Neither the former nor the latter problem are treated in this paper.

Generally, gossip protocols are scalable and very reliable; by randomly selecting peers, gossiping avoids message losses or node failures. Still, as noticed in [13], gossip schemes cannot deal with situations in which attackers falsify the information being disseminated from one peer to another, because gossip protocols do not verify the data being exchanged. This subclass of problems falls into the class of Byzantine faults. For this reason, there are a number of solutions trying to address these issues, and hereafter we will briefly discuss some of them.

Compared to previous structured overlay approaches, a more realistic solution (from the point of view of Byzantine faults) to P2P streaming is given by Dolev et al. in S-Fireflies [12]. The purpose of the P2P overlay network is double: tolerate Byzantine nodes and self-stabilize (to adapt dynamically to churn). S-Fireflies builds probabilistic graphs (random graphs) with nodes of low in- and out-degrees, that is stable in terms of Byzantine presence. The result of the algorithm is the enforcement of a ’rigid’ complete graph, so that nodes get to know all their neighbors (with a high probability). Onto this robust connection graph, Dolev et al. establish a monitoring mechanism that uses gossip to report node failures and propagate transmission rounds. The protocol is verifiable at the level of each node, so any peer can verify that another node communicates with correct neighbors. Moreover, there is an enforcement mechanism for nodes not to impersonate other nodes. In terms of the streaming session, there is a system-wide process with the purpose of updating all nodes in the network; even if a quarter of the nodes are temporarily faulty, the system is still able to recover.

The above solution provides some useful mechanisms for a P2P network to timely adapt to Byzantine faults: the construction of the random graph coupled with verifiable adaptiveness. However, although it controls the effect of malicious behavior in its general form, it does not deal with the causes of this behavior: nodes should be encouraged to participate in the game, because the more peers cooperate, the better the performance of the streaming session.

BAR-Gossip (Byzantine-Altruistic-Rational) [21] is the next step in making P2P streaming more secure and less treacherous. This new model increases the safety and liveness guarantees offered by Byzantine fault tolerance because it features an incentive-based mechanism for non-byzantine peers that may become malicious. This solution leverages on three different peer behaviors: purely byzantine, altruistic and rational nodes. The modification to the original peer selection scheme in gossip, is that this process is *pseudo-random and verifiable*. The strength of the overall protocol is due to several reasons:

- it is extremely robust when faced with Byzantine and selfish nodes,
- it can face collusion attacks,
- it provides stable short-term throughput, while the bulk of the other approaches target maximizing bandwidth on the long term,
- it is usable for short-window transmissions/streaming,
- it does not use reputations, so Sybil attacks are already dealt with.

BAR-Gossip functions in rounds, or transmission sessions; in every round, the source transmits the correct packets, and then peer nodes propagate these packets simultaneously in two schemes: a balanced exchange, and an optimistic push (of non-expired packets) protocol. BAR-Gossip also details the explicit exchange protocol between nodes, and takes all actions to balance the amount of information that is swapped between the two sides (very much like tit-for-tat). Moreover, the protocol seeks to monitor and reward/punish individual node activity so that there is an overall equilibrium between all nodes involved. Some solutions that this protocol gives to common problems, are:

- *Neighbor selection attacks*, because selection is verifiable. However, the convergence is not as fast as traditional gossip, because the mechanism of selecting neighbors replaces *sheer* randomness with pseudo-randomness **and** verification performed by the selected node onto the selector.
- *Nodes lying about their history*, because it is no longer desirable to lie in the short-term, because it

is no longer in their interest neither to under-report nor to over-report their packet history;
- *Forgery attacks* are repelled because of a clever mechanism by which data is first traded and verified, and then exchanged. If the data is forged, this would be noticed before the actual exchange, so the potential receiver would realize the attack and report it. Again, making sure that data is not forged prior to the exchange involves more overhead in the transmission;
- *Stability on the short term* is also grounded on the notion of Nash equilibria, so that any node would consider that its peers are following the protocol. This belief is actually an incentive for nodes, to abide by the protocol;
- *Free-riders* elimination is achieved by allowing junk updates, to compensate for the free-updates of an altruistic node.

Nevertheless, BAR-Gossip has its limitations. First if all, it only supports a static membership system, that is—all participating nodes must subscribe to the broadcaster before participating in any round—to this end, the system gains a centralized identity management scheme with a static list of node id-s. In the case of large amounts of nodes that come and leave, this could turn into a scalability problem. In addition, by using the comments in [2], the question of how would nodes discover themselves can arise; discovery is arguably the heart of gossip-based protocols, so a discovery solution should consider the topology of the network, and should be as decentralized as possible so that nodes can use it at all times.

Furthermore, it can be noticed that in the case of the optimistic push protocol, nodes are likely to waste bandwidth by sending junk—this again can turn into a problem if bandwidth is scarce or the quantity of junk that is being sent is large. From this points of view, an interesting idea would be that of Martin in [25], where the efforts are concentrated toward leveraging on altruistic nodes to carry the burden of rational nodes. In other words, in the BAR-Gossip solution, altruistic nodes and rational ones behave in the same way according to the specification; however, rational nodes may refuse to participate in some computation if the cost of their involvement is higher than their utility. In this case, performance of the overall system can be improved if altruistic nodes take upon themselves the work that was refused by the rational nodes. Of course, burdening altruistic nodes should be done with a reward, as much as 'selfishness' (rational nodes refusing participation) should be punished.

6 Discussion

There are a number of vulnerabilities that P2P streaming systems are prone to. When it comes to live streaming, problems get worse because of the bandwidth demand and timeliness of this type of systems. Seamless performance and attack-proof design are probably impossible to achieve at the same time. Even worse, given the large variety of attacks, countering all or most of them is even more challenging.

6.1 Tradeoffs: security vs. performance

BAR-Gossip is able to overcome a very large number of different attacks from the list in Section 4, but there are a number of trade-offs it has introduced to achieve this goal (see Table 7). The essential idea it applies in order to repel a large range of attacks is to *encourage nodes to behave*. If nodes misbehave, then they are punished; this can be easily implemented by some form of penalty or by placing the wrong-doers further away from the source of broadcast, thus ensuring that their possibility to harm is diminished. However, it is easy to notice that punishing one node may involve punishing the nodes that the current node is communicating with; thus, the effect of the punishment is likely to occur to innocent nodes as well. This is a trade-off in its own way: the decision to punish also some nodes that do not misbehave, in order to 'set an example' for other nodes.

Neighbor selection is another trade-off. Instead of allowing a tree-like structure in which nodes know from the beginning who to communicate with, it is wiser, from a security point of view, to sacrifice some performance in order to eliminate vulnerabilities as much as possible. Using the pseudo-random scheme together with selection verification is a far safer approach that using a centralized membership directory, which apart from bearing scalability problems, is also a single point of failure, not to mention the privacy issues it arises. From this point of view, there is another trade-off remarked by Jesi in [18]: BAR gossip is able to let neighbors control how random the peer selection process is for certain nodes, at the cost of ruling out dynamic peer membership. This is the reason why all peers first have

Table 7 Tradeoffs in BAR-Gossip

Tradeoff between ...	... and
punishing innocent nodes	fairness incentives
neighbor selection	dynamic peer membership
bandwidth utilization	allowing fake data delivery
timeliness	punishment for misbehaving nodes
performance	cryptographic schemes used

to register themselves to the broadcaster, before participating to the streaming system. Trading in the other direction, a system can admit dynamic membership, at the cost that nobody can control how randomly a node selects its neighbors.

Bandwidth utilization is the reason of another compromise. Since there can be nodes that offer packets (data) at a lower cost than any other nodes, this would imply that all requesters would crowd to use these free suppliers [7]; balance is brought into this scene by introducing the possibility that requester nodes receive junk if they turn into a burden for the altruistic ones. In this case however, bandwidth is wasted with the sole purpose of 'teaching a lesson' to the misbehaving nodes.

Timeliness of transmission is very seldom compromised in P2P streaming solutions. This is straightforward for the simple reason that users hate choppiness or low quality, and as long as they encounter any of them, they leave the system. Because this is not in the system's best interest—the more users, the higher the combined bandwidth—then timeliness is not a parameter to be touched. However, if nodes misbehave, punishment can take the form of placing these nodes farther from the source (if applicable), with the clear effect of obtaining packets which are closer to expiry.

Furthermore, protection mechanisms on the transmitted data ensure its integrity, confidentiality and fair-use. Overall, these mechanisms, ranging from signing, to briefcase negotiation and eventually briefcase exchange, function in the detriment of performance. Each node consumes bandwidth and processor cycles for the system's best interest, even if it is not always in its own interest. Of course, nodes should not be allowed to act by themselves, and as long as the risks that they encounter are the same for all other nodes, the same measures should be taken for them all. Needless to mention, countering all forms of 'anarchy' ensures the well-being of the entire system; thus an incentive/punishment technique needs to be in place to protect the streaming process.

6.2 Further work

Currently, collusion-based attacks remain one of the most problematic types of attacks in P2P streaming systems. Collusion do not necessarily mean the protocol is not respected, but it can also refer to a slight deviation from the protocol, which is hard to locate and cure. In BAR-Gossip, for instance, collusion may occur to rational nodes: a group of nodes that are not satisfied with the previous exchanges, group together in order to maximize collective utility. Their uncooperative behavior toward the rest of the network can manifest in a slower propagation of messages in the exterior of their group, compared to the one within the group. Again, this does not disrupt the overall protocol, it just decreases its effectiveness and jeopardize nodes equity.

More work is needed as to analyze the utility of nodes to deviate from the protocol; finding a bound for this utility, correlated with the application, would be useful in finding quantitative incentives for not deviating from the protocol. Moreover, churn in streaming systems remains another open problem, for which one possible solution can be that of self-adaptive networks, as described in Section 5.3.

7 Conclusions

There are two approaches to securing a peer-to-peer system: on the one hand, access control and identity management mechanisms can help to ensure that no malicious peers are allowed to join the system. This approach relies on the assumption that malicious behavior can be detected beforehand: peers refusing to comply with the rules of the protocol must not be accepted to join the system. On the other hand, peers can make promises they do not maintain afterwards. They can agree to be fair but eventually collude and unbalance the streaming process to their own advantage. In this second case, an audit-like mechanism can compensate the scheme. Observing what happens as the protocol is running can help an administrator determine if there are any system weak points that are being exploited. From this point of view, we think that a methodology is needed *to detect* that a P2P streaming system is under attack, and a study on what are the possible ways to compensate the damage.

In streaming (as in any other) systems, attacks occur because there is a vulnerability to be exploited. Once an attack happens, it needs to be confined to an area of the P2P network as small as possible. Once the attacker cannot easily gain control over a bigger portion of the system, some mechanisms are needed to detect the target and source of the attack. It is not always easy to detect who the malicious nodes are (they are always from within the network, assuming that no other hosts can interfere with the protocol). There are some techniques that can be used for this purpose: one of them requires a trust manager [30], with the limitation that the machine needs to be replaced periodically, and should not be central to the whole network (since we want to eliminate single points of failures). A complementary approach is to use the mechanism of incentives and punishments, where nodes are stimulated to stick to the protocol; if they do not comply, then a distributed monitoring mechanism (performed via the malicious neighbors of a node) should help enforcing a punishment onto the bad performers. This incentive and punishment approach is so far the only distributed mechanism able to offer guarantees that on the long run, peer nodes will comply with the rules of the system.

As discussed in the paper, tree-based streaming in P2P networks are not only vulnerable to protocol failure, but are also far faster contaminated by an attacker if the hierarchy of nodes is fixed. In the eventuality no other constraints are put onto the degree of each node (either inbound or outbound), then the structure is vulnerable and cannot contain most attacks. This happens because in a tree, if a node is contaminated, then its children will be too. In a mesh, on the other hand, the infection spreads in a one-by-one fashion rather than in a one-to-many fashion.

Compared to tree-based streaming, mesh and gossip approaches are more robust, scalable and Byzantine-tolerant. The largest amount of recent works concentrate on either of these two approaches, and attach a wide variety of additional mechanisms in order to counteract as many attack types as possible. The overall trend is to delegate many monitoring and security functions to each peer instead of keeping separate entities exclusively for these tasks. Membership and neighbor selection mechanisms are driving the flow of any different protocol, while tune-ups mostly try to leverage churn and node coalitions.

Acknowledgement This work is supported by the European Commission through the NAPA-WINE Project (Network-Aware P2P-TV Application over Wise Network—www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant no. 214412, and by the Italian MIUR PRIN Project ‘Autonomous Security’.

References

1. Slashdot (2007) Skype blames Microsoft patch Tuesday for outage. http://slashdot.org/articles/07/08/20/150258.shtml
2. Alvisi L, Doumen J, Guerraoui R, Koldehofe B, Li H, van Renesse R, Tredan G (2007) How robust are gossip-based communication protocols? SIGOPS Oper Syst Rev 41(5): 14–18
3. Bianchi G, Bonola M, Falletta V, Proto FS, Teofili S (2008) The sparta pseudonym and authorization system. Sci Comput Program 74(1–2):23–33
4. Camenisch J, Lysyanskaya A (2001) An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Proc of the int conference on the theory and application of cryptographic techniques (EUROCRYPT ’01). London, UK, Springer-Verlag, pp 93–118
5. Carra D, Lo Cigno R, Biersack EW (2007) Graph based analysis of mesh overlay streaming systems. IEEE J Sel Areas Commun 25:1667–1677
6. Carra D, Lo Cigno R, Biersack EW (2008) Stochastic graph processes for performance evaluation of content delivery applications in overlay networks. IEEE Trans Parallel Distrib Syst 19:247–261
7. Ciullo D, Mellia M, Meo M, Leonardi E (2008) Understanding P2P-TV systems through real measurements. In: Proc of the IEEE global telecommunications conference (GLOBECOM’08)
8. Conner W, Nahrstedt K (2007) Securing peer-to-peer media streaming systems from selfish and malicious behavior. In: MDS ’07: Proc of the 4th on Middleware doctoral symposium. ACM, New York, pp 1–6
9. Conner W, Nahrstedt K, Gupta I (2006) Preventing DoS attacks in peer-to-peer media streaming systems. In: Proc of the 13th annual conference on multimedia computing and networking (MMCN’06), San Jose
10. Dabek F et al (2001) Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service. In: Proc of the 8th workshop on hot topics in operating systems (HotOS), Schloss Elmau
11. Dhungel P, Hei X, Ross KW, Saxena N (2007) The pollution attack in P2P live video streaming: measurement results and defenses. In: Proc of the 2007 workshop on peer-to-peer streaming and IP-TV (P2P-TV’07). ACM, New York, pp 323–328
12. Dolev D, Hoch EN, van Renesse R (2007) Self-stabilizing and byzantine-tolerant overlay network. In: Tovar E, Tsigas P, Fouchal H (eds) Proc of the 11th int conference on principles of distributed systems (OPODIS’07), LNCS, vol 4878. Guadeloupe, French West Indies. Springer, New York, pp 343–357
13. Han K, Pei G, Ravindran B, Jensen E (2008) Real-time, byzantine-tolerant information dissemination in unreliable and untrustworthy distributed systems. In: Proc of the IEEE int conference on communications (ICC’08). pp 1727–1731
14. Haridasan M, van Renesse R (2006) Defense against intrusion in a live streaming multicast system. In: Proc of the 6th IEEE int conference on peer-to-peer computing (P2P’06). IEEE Computer Society, Cambridge, pp 185–192
15. Haridasan M, van Renesse R (2008) SecureStream: an intrusion-tolerant protocol for live-streaming dissemination. Comput Commun 31(3):563–575
16. Jelasity M, Montresor A, Babaoglu O (2009) T-Man: gossip-based fast overlay topology construction. Elsevier Comput Networks 53:2321–2339
17. Jennings C, Lowekamp B, Rescorla E, Baset S, Schulzrinne H (2009) REsource LOcation And Discovery (RELOAD) v. 6, P2PSIP Internet-Draft, IETF. http://tools.ietf.org/html/draft-ietf-p2psip-base-06. Accessed 9 Nov 2009
18. Jesi G (2006) Secure Gossiping Techniques and Components. PhD thesis, University of Bologna, Dept of Computer Science
19. Kermarrec A-M, van Steen M (2007) Gossiping in distributed systems. SIGOPS Oper Syst Rev 41(5):2–7
20. Leonardi E, Mellia M, Horvath A, Muscariello L, Niccolini S, Rossi D (2008) Building a cooperative P2P-TV application over a wise network: the approach of the European FP-7 strep NAPA-WINE. IEEE Commun Mag 46(4):20–22
21. Li HC, Clement A, Wong EL, Napper J, Roy I, Alvisi L, Dahlin M (2006) BAR gossip. In: Proc of the 7th SIGOPS symposium on operating systems design and implementation (OSDI’06). USENIX Association, Seattle, WA
22. Liu Y, Guo Y, Liang C (2008) A survey on peer-to-peer video streaming systems. Peer-to-Peer Networking and Applications 1(1):18–28. http://www.springerlink.com/content/c62114g6g4863t32

23. Magharei N, Rejaie R (2007) PRIME: peer-to-peer receiver-driven mesh-based streaming. In: Proc of the 26th IEEE int conference on computer communications (INFOCOM'07). IEEE, pp 1415–1423
24. Magharei N, Rejaie R, Guo Y (2007) Mesh or multiple-tree: a comparative study of live P2P streaming approaches. In: Proc of the 26th IEEE int conference on computer communications (INFOCOM'07), pp 1424–1432
25. Martin J-P (2007) Leveraging altruism in cooperative services. Technical Report TR-2007-76, Microsoft Research, Cambridge
26. Rowstron A, Druschel P (2001) Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: Proc of the 18th int conf on distributed Systems Platforms, Heidelberg
27. Seedorf J, Burger E (2009) Application-layer traffic optimization (ALTO) problem statement. RFC 5693, IETF
28. Seibert J, Zage D, Fahmy S, Nita-Rotaru C (2007) Experimental comparison of peer-to-peer streaming overlays: an application perspective. Technical Report CSD TR 07-020, Purdue University
29. Seibert J, Zage D, Nita-Rotaru C (2008) Won't you be my neighbor? Neighbor selection attacks in mesh-based peer-to-peer streaming. Technical Report, Purdue University
30. Shetty S, Galdames P, Tavanapong W, Cai Y (2006) Detecting malicious peers in overlay multicast streaming. In: Proc of the 31st IEEE conference on local computer networks (LCN'06), Florida
31. Silverston T, Fourmaux O (2006) Source vs data-driven approach for live P2P streaming. In: Proc of the int conference on networking, int conference on systems and int conference on mobile communications and learning technologies (ICNICONSMCL '06), IEEE Computer Society, Washington, DC, p 99
32. Singh A, Castro M, Druschel P, Rowstron A (2004) Defending against eclipse attacks on overlay networks. In: Proc of the 11th workshop on ACM SIGOPS European workshop, p 21
33. Singh K, Schulzrinne H (2005) Peer-to-peer internet telephony using SIP. In: Proc of the int workshop on network and operating systems support for digital audio and video (NOSSDAV'05). ACM, Stevenson, pp 63–68
34. Wallach DS (2003) A survey of peer-to-peer security issues. In: Okada M, Pierce BC, Scedrov A, Tokuda H, Yonezawa A (eds) Proc of the Mext-NSF-JSPS int symposium on software security—theories and systems (ISSS'02), LNCS, vol 2609. Springer, Tokyo, pp 42–57
35. Yang J, Li Y, Huang B, Ming J (2008) Preventing DDoS attacks based on credit model for P2P streaming system. In: ATC '08: Proc of the 5th international conference on autonomic and trusted computing. Springer, Berlin, pp 13–20
36. Yang S, Jin H, Li B, Liao X (2009) A modeling framework of content pollution in Peer-to-Peer video streaming systems. Comput Networks 53(15):2703–2715
37. Zhang X, Liu J, Li B, Yum Y-S (2005) CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In: Proc of the 24th IEEE int conference on computer communications (INFOCOM'05), vol 3, pp 2102–2111
38. Zhou M, Liu J (2005) A hybrid overlay network for video-on-demand. In: Proc of the IEEE int conference on communications (ICC'08), pp 1309–1311

Gabriela Gheorge is a second year PhD student at ICT Doctorate School Trento. The focus of her research lies with the enforcement of security policies in distributed environments. Other interests include access and usage control within the same environments, in Web services and Service-Oriented Architectures. She graduated at "Politehnica" University of Bucharest with a thesis dealing with security functionality in Service-Oriented Architectures; the thesis was carried out the Universitaet der Bundeswehr Muenchen, Germany.

Renato Lo Cigno is Associate Professor at the DISI Department of the University of Trento, Italy, where he is one of the founding members of the Networking research group. He received a Dr. Ing. degree in Electronic Engineering from Politecnico di Torino in 1988. From 1989 to 2002 was been with the Telecommunication Research Group of the Electronics Department of Politecnico di Torino. From June 1998 to February 1999, he was at the CS Department at UCLA as Visiting Scholar under grant CNR 203.15.8. He is coauthor of more than 120 journal and conference papers in the area of communication networks and systems. His curent research interests are in design and performance evaluation of wired and wireless networks, including optical switching networks and overlay, peer-to-peer systems; simulation techniques and modeling; and resource management and congestion control. Renato Lo Cigno is member of the IEEE and ACM; he is Area Editor of Computer Networks (Elsevier), and has served as Chair or TPC member of several leading conferences, including IEEE INFOCOM, Globecom, ICC, IEEE/ACM MSWiM, IEEE MASS.

Alberto Montresor is Associate Professor of Computer Science at the University of Trento, Italy. He received his Ph.D. in 2000 from the University of Bologna, Italy, where he designed Jgroup, a partition-aware group communication system. He served as Research Associate in Bologna until 2005, when he moved to Trento. He is author of more than 50 papers in international conferences and journals, and he has been active in several European projects in the field of distributed computing and complex adaptive systems.