16. Self-Organization in Peer-to-Peer Systems

Hermann De Meer, Christian Koppen (University of Passau)

Self-organization is seen as an attractive feature of Peer-to-Peer networks although meaning and significance of this term are far from being clear. In this chapter, principles of self-organization in Peer-to-Peer systems are identified. The potential enabled by incorporating these principles and further potentials of increasing the degree of self-organization are outlined. The Active Virtual Peer (AVP) concept is used as an example for incorporation of an enhanced level of self-organization into Peer-to-Peer systems.

16.1 Introduction

In the year 1999, the first Peer-to-Peer system, Napster [436], began its (short) career in the Internet. The popularity of Peer-to-Peer networks has grown immensely ever since. Nowadays, the traffic load on the Internet appears to be dominated by Peer-to-Peer applications (see Chapter 22 for details). As the downside of the success story scalability and flexibility issues became visible. If well understood and carefully implanted self-organization may provide a useful means to handle these challenges. But since self-organization may resist imposed control if done naively, self-organization can as well be the source of inefficiency. Many Peer-to-Peer systems have been advertised as being self-organizing, although meaning and significance of this claim are far from being clear. There are several classes of Peer-to-Peer systems that exhibit different properties with different degrees of self-organization. Peer-to-Peer systems have to provide services like routing, searching for and accessing of resources. An open question is if and how much can self-organization, with all its illusiveness, emerge as an essential means for improving the quality of the services. Improved service quality, thereby, is to be achieved equally for performance, robustness, security and scalability in an all open world.

Based on the characteristics as outlined in Chapter 15, we describe criteria for self-organization of Peer-to-Peer systems. In Section 16.2.1, these criteria are first introduced and motivated. Following that, the criteria are applied to some of the more popular unstructured and structured Peer-to-Peer systems in Section 16.2.2 and Section 16.2.3, respectively. In each case the overall degree of self-organization incorporated is first identified and then potential enhancements of self-organization are discussed. In Section 16.3 the Active Virtual Peer concept is introduced as an example for a higher degree of self-

R. Steinmetz and K. Wehrle (Eds.): P2P Systems and Applications, LNCS 3485, pp. 247-266, 2005.

organization in Peer-to-Peer systems from certain perspectives. Section 16.4 concludes this chapter.

16.2 Evaluation of Peer-to-Peer Systems

16.2.1 Criteria

The analysis of Peer-to-Peer networks is based on the characteristics of self-organization as introduced in Chapter 15. Since these characteristics still remain somewhat elusive we first provide more specific criteria that can be easier applied for our purpose. The goal is to use simpler criteria for the analysis of the extent to which a Peer-to-Peer system can be characterized as being self-organizing. We divide the criteria into two groups: *basic criteria* and *criteria for autonomy*. A conformity with criteria for autonomy enables a system to adapt autonomously and develop a "life of its own". The criteria to be investigated are the following:

Basic criteria

- **Boundaries:**
  The boundaries of a self-organizing Peer-to-Peer system should be self-determined. In other words, the decision about the affiliation of its components, i.e., the peers, is should be made by the system itself. The boundary of a Peer-to-Peer system can be understood as the point where new, unknown nodes enter the system. Peers which offer new nodes the possibility to join are often called *bootstrap* nodes.
- **Reproduction:**
  A self-organizing Peer-to-Peer system can and does reproduce its structure. This may include addition, removal or change of a peer, its data or its relations or connections to other peers. Reproduction does not necessarily mean the creation of an isomorphic copy but may include mutations.
- **Mutability:**
  A self-organizing Peer-to-Peer system is able to change its structure. The change may concern constitution and number of peers and relations. Possible instances are the change of connections or the formation of clusters.
- **Organization:**
  A self-organizing system is organized in form of a hierarchy, a heterarchy or both. The organization has effects on the system's structure, e.g., whether there are fixed communication paths or single points of failure.
- **Metrics:**
  A self-organizing system is able to detect perturbations triggered from the environment. For Peer-to-Peer systems, the following perturbations are typical:

  - Failure of a peer or connection,
  - Overload or DoS-attacks,
  - Manipulation of data ("fakes").

  In addition, Peer-to-Peer systems often suffer from the problem of "freeriders" – peers which make use of the system without supplying resources. Effects of freeriders are not considered as perturbations because they have their cause *within* the system.
- **Adaptivity:**
  A self-organizing Peer-to-Peer system is able to react to perturbations appropriately. The reaction may include restructuring of peers, authorization of peers to avoid "fakes" and incorporation of redundancy as a further measure of precaution.

Criteria for autonomy

- **Feedback:**
  A self-organizing Peer-to-Peer system is often exposed to positive and negative feedback, whereupon the structure or behavior of the system changes in a balanced way. Feedback includes messages that peers send to each other.
- **Reduction of complexity:**
  A Peer-to-Peer system which is self-organizing develops structures and hides details from the environment to reduce the overall complexity. This may include the forming of clusters or the creation of other entities, e.g., an Active Virtual Peer (AVP) (described in Section 16.3) or a holon which is a *group* of agents that appears as a *single* agent to the outside (the concept of holons is further detailed in [208] and [270]).
- **Randomness:**
  A self-organizing system makes use of randomness as a prerequisite for creativity. This allows the creation of new structures with little effort. An example from another discipline are ant algorithms where ants decide randomly between different paths when they have no sufficient knowledge about their environment (see Section 15.4.2 for details).
- **Self-organized criticality (SOC):**
  A system which is self-organizing drives itself into a state of criticality. Too much order as well as too much disorder are to be avoided by adequate procedures. This should result in an increased degree of flexibility because the system is able to cope with different types of perturbations.
- **Emergence:**
  A self-organizing Peer-to-Peer system shows properties that no single peer has on its own, or properties that may have been unknown at design time. Peers forming a small-world exemplify an emergent structure.

Besides the degree of conformance to these criteria, every system has an **identity** (in the meaning described in Section 15.3.3), or a main purpose, that is an essential characteristic of the system. The identity of a Peer-to-Peer

system is (as usual for information systems) imposed from the outside, i.e., from the developers, and does not arise self-determined.

The results of our analysis are presented in Tables 16.1 and 16.2. Explanations are given in Sections 16.2.2 and 16.2.3. We use the following structure for each subsection of Sections 16.2.2 and 16.2.3: the first paragraph gives a very short description of the analyzed Peer-to-Peer system (for details, we refer to other chapters of this book). The second paragraph contains our results concerning the basic criteria identity, boundaries, reproduction, mutability & organization. The third paragraph deals with the basic criteria metrics & adaptivity. The last paragraph illustrates conformance to the criteria for autonomy.

16.2.2 Unstructured Peer-to-Peer Networks

In this subsection, we analyze some of the more popular Peer-to-Peer systems for their ability to self-organize.

	Napster	Gnutella	FastTrack	eDonkey	Freenet
Identity	Filesharing				Anonymity
Boundaries	×	×	◯	◯	×
Reproduction	×	×	×	×	✓
Mutability	×	×	◯	◯	×
Organization	×	×	✓	✓	×
Metrics	◯	◯	◯	◯	◯
Adaptivity	◯	×	◯	◯	✓
Feedback	×	×	×	×	✓
Reduction of complexity	×	✓	×	×	×
Randomness	×	×	×	×	×
SOC	×	×	×	×	×
Emergence	×	✓	×	×	×

Table 16.1: Self-organization in unstructured Peer-to-Peer systems. The symbols show the degree of conformance with the criteria listed in Section 16.2.1.
✓ – full conformance ◯ – partial conformance
× – no conformance

Napster

Napster [436] was conceived as a platform to share audio data in the well-known MP3-format.

The server of the system is the only bootstrap node. It admits every peer to enter, so boundary conditions are not actively enforced by the system itself. Of course there could be set external policies for admittance, but these policies are not an integral part of the Peer-to-Peer system. Consequently, one of the essential characteristics of self-organization, namely self-bounding, is not fulfilled. Neither peer structure nor data is actively reproduced; when a peer leaves the system, its data is no longer available if not provided by other peers. Clients cannot take over management tasks from the server and the server does not share files. As a result, mutability is not given as far as the system structure is concerned. The type of organization is a mix of a very flat hierarchy (between the clients and the server) and a heterarchy (among the clients). The heterarchical organization is of advantage – it does not affect the whole system if a single peer fails. Unfortunately, clients direct their search requests to the central server only, which is therefore urgently required for the operation of the system. Thus the server is a single point of failure (SPoF) which ruins the positive effects of the heterarchy.

Metrics can be attained by means of keep-alive-messages (`ping/pong`) that peers exchange among each other; these messages are an appropriate way to detect the failure of a node or connection. If such a message fails to appear, the server can be asked once again for the respective file to be located at another peer. Such a form of adaptivity can be of advantage to the overall system. However, no explicit precaution against or response to overload conditions is taken into account. Similarly no defence against possible DoS attacks or against infiltration by corrupted data is integrated. Thus, the criterion of metrics is not fully satisfied.

Keep-alive-messages are a form of internal communication which is an indication for feedback. But for conformity to criterion "feedback", reactions and structural changes are also necessary. This is not the case in Napster, so feedback is hardly incorporated. No further properties that satisfy the criteria for autonomy are known.

Gnutella

Gnutella [250] was developed to avoid effects of centralization (including limitations of scalability, unused resources on the clients and the server as SPoF). In this section, we refer to version 0.4 of Gnutella, where every peer ("servent") is equal.

Since every servent is equal every servent is a bootstrap node, too. In analogy to Napster, no node is barred from accessing the system, so the boundaries are not determined by the system. Also, no automatic reproduc-

tion occurs. A difference to Napster lies in the (strictly) heterarchical organization of the peers. This causes search and signaling being done in form of flooding which has been identified as the main culprit for limitations in scalability [518]. Since the heterarchical organization remains invariant, no restructuring can occur and thus no mutability is given.

Gnutella uses the same mechanism to achieve metrics as in Napster, peers excessively exchange keep-alive-messages. The risk of overloading is notably high due to flooding of messages through the network; there are no provisions against manipulation of data. Concerning adaptivity, Gnutella resembles Napster: a missing `pong` causes the peer which sent the `ping` to terminate related connections. But since the network is flooded with keep-alive messages, adaptivity is even harder to achieve than in Napster.

In analogy to Napster, keep-alive messages are used. But besides the termination of connections no structural changes occur, thus feedback control is not incorporated. An interesting property of a Gnutella network is the connectivity pattern or node degree. Very often the node degrees can be approximated by a power law distribution [517]. Gnutella additionally seems to feature the emergent property of a small-world network. This can be explained by the way the system is used: a few peers have high capabilities (bandwidth and capacity) and provide a lot of files. This is accompanied by many peers primarily connecting to such a privileged servent. Many peers only download something and then disconnect without having offered anything ("freeriders"). This structure is typically formed without external influence and is facilitated by the flexibility of the underlying heterarchy. The traffic is strongly controlled by the servents with high capabilities, which reduces stress on the network resources.
The existence of randomness or an appearance of self-organized criticality cannot be attributed to Gnutella according to the definitions as used throughout this chapter.

FastTrack

FastTrack [202] can be seen as a hybrid of Napster and Gnutella. All peers are equal, but every peer can decide to become a *SuperNode*, which offers services to other peers ("successors") that connect to it. This concept is a structural response to Gnutella's usage profile.

The boundaries of FastTrack can at least be somehow influenced, because the system offers a possibility to limit bandwidth and connection count. When overload is on the raise, a peer can refuse requests (including requests from new nodes that want to enter the network). This feature, however, has to be adjusted manually and is not self-organizing; furthermore it does not allow to reject a request or node based on some characteristics. Thus FastTrack is not fully compliant to the criterion "boundaries". FastTrack clients do not replicate data without user interaction, so no active reproduction occurs. The

SuperNode concept is a form of mutability because it allows differentiation of peers, albeit not automatically. The SuperNode concept allows for a dynamic (but manual) adjustment of the organizational form; besides the underlying heterarchy a (very flat) hierarchy between a SuperNode and its successors can emerge. A drawback of this implementation is that every node is exclusively assigned to one SuperNode (which makes it a local SPoF, a common problem in hierarchies).

FastTrack uses the same mechanisms as Napster to obtain metrics. If a `pong`-message of a SuperNode is missing its successors connect to a new SuperNode; thus a perturbation may affect the performance of a system but not its basic operation. Such a characteristics can be seen as an indication of a higher level of adaptivity. Another enhanced technique of dealing with overload situations is "swarming": a single peer can download different parts of the same file from various peers simultaneously. The problem of "fakes" is not addressed and could pose a major problem for such a network.

Another newly introduced concept is the "rank" which determines the priority of a peer. The inclusion of this parameter prioritizes the traffic flows. But since it does not change the traffic flows (which would be a change of the structure of the system) this is no form of feedback in the sense of Section 16.2.1. One could argue that the SuperNode concept is an indication for the reduction of complexity; but the change of the peer status must be done manually and is non self-organizing. No further properties were found to conform a criterion of autonomy.

eDonkey

The eDonkey network [185] has strong parallels to FastTrack, but is geared to the transfer of very large files. In addition to swarming, peers can help each other by means of "hording", i.e., swapping received data among each other so that data does not need to be downloaded from (far away) sources. In eDonkey, a peer runs either a client application or a server application or both.

eDonkey conforms to the "boundaries" criterion in analogy to FastTrack: every interested node can connect to a server, while the server may reject requests if it is overloaded. Also in eDonkey, no reproduction is done automatically. Every peer (preferably with high capabilities) can run a server. A server in eDonkey is comparable to the SuperNode concept in FastTrack, thus the same arguments are valid concerning mutability. The organization of peers is quite similar, too; as an enhancement, a client can connect to multiple servers at the same time, which makes the system more robust.

Considerations about metrics go according to the Peer-to-Peer systems analyzed above. The failure of a node does not concern the whole network; if the node ran a server, its clients can connect to another server and continue. Swarming and hording are designed to reduce the traffic load. The use of

native UDP-transport could pose a problem due to the lack of congestion, error or flow control. Traffic flows to servers which are offline can continue for a long time, wasting resources. eDonkey is the first Peer-to-Peer system that addresses the problem of "fakes". For this purpose, there are websites which list files with probably incorrect or inconsistent data. This allows the *manual* circumvention of fakes – so no self-organization here.

There are mechanisms to handle priorities and queuing; these have, in analogy to FastTrack, no impact on the structure of the system, but merely reorder the traffic flows. Thus, feedback cannot be detected. Besides, there's no indication for properties that fulfill one of the criteria for autonomy.

Freenet

The identity of the Freenet approach [231] differs from the one of the systems covered so far. Its purpose is to provide an infrastructure for free and anonymous information exchange. The detailed mechanisms are described in [123] and [124].

Like the other approaches, Freenet offers no control for joining or leaving nodes, and thus has no means to decide about its boundaries in a self-determined way. But it provides the reproduction of data: requested information is cached on the nodes between source and target. This results in the movement of data towards its requesters. More than that, it leads to the duplication of popular data while unrequested data is timing out. The data is adapted to user requests. All peers are equal; they are organized in form of a heterarchy which does not allow for mutability of the system structure. In addition, every node only knows a fixed number of neighbors, which leads to inefficiency on the one hand but higher potential for anonymity on the other hand.

The use of metrics, especially the handling of perturbations, is exceptionally interesting in Freenet, as it is hard for perturbations to have an effect on the system at all. A failure of a peer or connection can be tolerated, because its data (at least the popular part of it) is cached on its neighbors. This is also why overload of a node is unlikely to occur: the more peers request data that a certain node holds, the more copies will be made and, thus, later requests will not even reach the original node but be answered by increasingly "closer" nodes. A similar argument applies to the threat of DoS attacks – the intended effect is a temporary high load in the network until an attacker gets swamped by responses. All these properties result from Freenet having the focus on *data*, and not on the peers. An attack or request can not be done to a peer, but to data (which is adapted on demand). The manipulation of data is also hard to achieve because of the multiple encryption techniques and the lack of knowledge about the location of data. As a conclusion, Freenet in fact does offer less *measures* and *reactions* to perturbations but incorporates *preventions* by design.

The adaption of data to user requests is a form of feedback: favored data is "amplified" while unpopular information is deleted. Further conformance to the criteria of autonomy could not be detected.

Note: All mentioned indications of self-organization in Freenet concern the *data* only, and not the structure (which would be necessary for more complete self-organization).

16.2.3 Structured Peer-to-Peer Systems

Structured Peer-to-Peer networks make use of distributed hash tables (DHT) to allow for a more efficient allocation of resources and routing to information.

	Chord	PAST	CAN	NICE
Identity	Allocation	Storage & allocation	Allocation	Distribution
Boundaries	×	×	×	×
Reproduction	○	○	○	○
Mutability	○	○	×	✓
Organization	✓	✓	○	○
Metrics	○	○	○	○
Adaptivity	○	✓	○	○
Feedback	✓	×	×	×
Reduction of complexity	×	×	×	✓
Randomness	×	×	×	×
SOC	×	×	×	×
Emergence	×	×	×	×

Table 16.2: Self-organization in structured Peer-to-Peer systems. The symbols show the degree of conformance with the criteria listed in Section 16.2.1
✓ – full conformance ○ – partial conformance
× – no conformance

Chord

Chord [117], [575] arranges all peers on a ring of size N. Every peer holds a routing table which contains log N "fingers", i.e., addresses of other peers.

These are not set arbitrarily but in a way to gain small-world characteristics: there are many entries for nearby peers and a few for distant ones. Due to the universality of DHTs, the identity of Chord is not filesharing but allocation of data in a more general way.

In analogy to Gnutella, every peer can serve as a bootstrap node and every unknown node is admitted to the system, so the "boundaries" criterion is not met. Chord does not reproduce peers or connections, but the ring structure is always preserved. Additionally, the redundant storage of data is possible: when a new node enters the system, it obtains the data it is responsible for while its predecessor may keep a copy of (at least a part of) it. In case of the departure of a node, its neighbors redistribute the data among each other. So Chord conforms at least partially the "reproduction" criterion. Since the ring structure is a design principle it is immutable. However, a smart (or maybe dynamic re-) assignment of `ID`s can be used to balance the traffic load so that mutability is partially given. Concerning the organization, arbitrary connections may exist between peers, so peers form a heterarchy. Nevertheless, communication paths are not chosen arbitrarily but structured because every peer has a routing table. This table is different for each peer so that no SPoF exists. This means that every peer is part of *many* hierarchies (i.e., routing tables with distance as order). Taken altogether, Chord implements a sophisticated combination of hierarchy and heterarchy.

Metrics are used as in unstructured Peer-to-Peer systems, the failure of a node can be detected by the absence of keep-alive-messages. If such a failure occurs, the predecessor of the missing node takes over the responsibilities of its successor. This allows a high measure of robustness in conjunction with the application of redundancy. On the other hand, an overload of peers is not unlikely: since data is portioned in disjoint parts, there is only *one* peer for every piece of information. Thus, the nodes which keep popular data are at high risk to be congested. The problem of intentionally faked data or routing tables has not been addressed so far. Taking it all together, Chord offers some mechanisms to support adaptivity.

The peers' routing tables are periodically checked for consistency. This is a form of feedback which is explained as follows. Entry i in the routing table of peer n contains the address of a peer p whose distance to n on the ring is between 2^i and 2^{i+1}. If p fails, n searches for a peer q which is neighbor of p and whose distance to n also is between 2^i and 2^{i+1}. If such a peer q is found, n directs the respective requests to q instead of p. Since this means a change of connections, the system structure is changed (the ring is stabilized). Thus, messages between peers lead to a more stable system structure which preserves consistency and efficiency and conforms to the criterion "feedback". There is no indication for the satisfaction of another criterion for autonomy.

PAST

PAST [476] is an approach for the allocation and archival storage of data. The allocation is managed by Pastry [527], an algorithm based on prefix routing. Each of the b^n peers in a Pastry system is part of n nested clusters and knows $b-1$ peers on each cluster level which resembles a scale-free network. Further details can be found in [526], [528].

Every interested peer gains access to the system, so the boundaries are not exclusively determined by the system. In analogy to Chord, replication is possible; the actual deployment can be adjusted by the replication parameter k on a per-file basis. Since this parameter is an integral part of PAST, at least reproduction of *data* can be identified. The system is immutably structured in form of a $b*$-tree with peers being the leaves. But as in Chord, a dynamic assignment of `IDs` could be used for load-balancing. Another analogy to Chord is the organization. Peers may arbitrarily connect to each other (heterarchy), but communication is forced to traverse along the cluster hierarchy. Additional data structures named `leaf set` and `neighborhood set` allow direct and thus efficient communication to topological or domain specific *nearby* peers. The cluster hierarchy on the other hand offers an upper bound to the number of necessary hops to deliver a message to distant peers. Pulled together, the organization is designed to support for both efficiency and robustness.

Concerning metrics, PAST differs only slightly from aforementioned Peer-to-Peer systems – the number of keep-alive-messages sent to nearby peers is higher than those sent to distant peers. When a failure or overload occurs, redundancy is used to confine effects locally so the global system is safeguarded. The replication parameter, which is crucial to support adaptivity, is adjusted manually only so that self-organization is limited in that respect. PAST offers a dedicated security concept that allows authorization via *smartcards* after a peer has joined. This clearly reduces the danger of fakes (that represent a negative perturbation), but on the other hands restricts strongly the application of the system and number of users.

No property was found to satisfy one of the criteria for autonomy.

CAN

With the approach of content addressable networks (CANs) [504] data is organized in form of D-dimensional vectors; for every dimension a different hash function is used. Requests are routed from a node to both of his neighbors in every dimension, what leads to a complexity of $\mathcal{O}(D \sqrt[D]{N})$ with constant storage cost for each of the N peers.

As in the other cases, no peer willing to enter the system is rejected and thus the boundaries are not exclusively determined by the system itself. First ideas considering reproduction of data (similar to Chord) are described

in [505]. Peers are immutably arranged in form of a D-dimensional torus, so mutability in the sense of section 16.2.1 is excluded by design. Peers form a hierarchy because every peer communicates only with its direct neighbors. If failure of a peer p is detected, p's neighbors correct their neighborhood information so that the system stabilizes. Therefore, the occurrence of a SPoF can be excluded in this case.

Peers use keep-alive messages to build up metrics (they check if their neighbors are available). Whenever a failure of a peer p is detected, one of its neighbors takes over p's data range; a proper defined replication scheme could lead to a high degree of adaptivity in this case. It is noticeable that the number of affected nodes in the case of failure is constant, since the failed peer has only influence on its direct neighbors. The problem of "fakes" is not discussed.

Characteristics that conform to the criteria for autonomy could not be identified.

NICE

NICE [449] is a framework for group cooperation such as multicasting. Peers are organized in form of a b-tree (which is kept in balance via merge- and split-mechanisms). A *cluster* is a group of peers on the same level of the b-tree. A cluster has a designated node called *leader* which represents the (exclusive) communication interface between the children and the rest of the system. All leaders of clusters on level i in the b-tree form a new cluster on level $i+1$. NICE can therefore be seen as a hierarchy of leaders with "normal" peers as leaves. A cluster is able to change its leader but the change process may be expensive in terms of management cost.

A joining peer is assigned to the most appropriate cluster (depending on a definable metrics, e.g., the `RTT`) which means that some characteristics of the new peer are taken into account for the join process. But since these characteristics are only used to decide *where* and not *whether* a new peer is added, the system has few control about its boundaries. One of the main goals of NICE is efficient distribution of data which implies reproduction of data; peers or connections are not reproduced, though. Due to the incorporation of split- and merge-mechanisms, NICE is able to change the system structure which is an indication for mutability. The organization is hybrid because peers within a cluster are fully meshed and can communicate directly which means that (small) heterarchies are encapsulated within the (big) hierarchy of leaders. This hybrid form of organization reveals a possible problem of NICE. Since peers within a cluster can communicate with the rest of the system only via the cluster leader, the leader is a SPoF. An enhancement of NICE, called ZigZag [595], confirms this: the leaders are supported by *vice-leaders* which exclusively care about the distribution of data, while the leaders are responsible for management only.

NICE uses keep-alive ("heartbeat") messages to build up metrics concerning perturbations. The failure or attack of a peer can have system-wide effects if the target of an attack is a leader. But due to highly meshed clusters failures are detected quite fast, the peers then simply elect a new leader. Clusters formations as well as leaders may be changed so that NICE can adapt to perturbations. There are no measures to detect or avoid manipulation of data.

The technique of clustering is an indication for the reduction of complexity. A cluster is constituted of multiple components, but appears as one entity to the outside. Additionally, communication follows clearly defined rules (traversing the b-tree). The obedience to rules is another indication for a reduction of complexity (as described in Section 15.3.10). Clustering as incorporated in NICE resembles the concept of "holons" which is further described in [208] and [270].

16.2.4 Summary of Peer-to-Peer Evaluations

None of the analyzed Peer-to-Peer systems can be called fully *self-organized* in terms of the criteria described in Section 16.2.1. In this section some of the advantages and disadvantages of incorporating the principle of self-organization into Peer-to-Peer networks are discussed.

Advantages Gained from Using Self-Organization

The ability of NICE to change the system's structure dynamically allows adaptivity and efficient communication. Gnutella's pure heterarchical design gives enough freedom to cultivate a small-world structure. The DHT approaches combine a "physical" [1] heterarchy with different levels of logical hierarchies; this leads to the situation that no peer is more important than any other, at the same time allowing for efficient communication between arbitrary peers, that is, in $\mathcal{O}(\log n)$ in most approaches. The reproduction of data, especially in Freenet but also in the DHT approaches, increases the robustness of a system significantly. FastTrack can reject new nodes if it is congested, NICE evaluates new peers – first steps to build and control the boundaries of a system. The fake lists developed for eDonkey are at least an attempt to anticipate the manipulation of data.

Disadvantages Implied from Using Self-Organization

None of the investigated Peer-to-Peer networks have the ability to decide completely in a self-determined way about the respective boundaries, which

[1] at least as physical as an overlay network can be

evocates the problem of freeriders and attackers. Even worse, most systems even have no metrics to detect perturbations appropriately, not to mention reactions to perturbations. While Napster and NICE can be taken as examples for the power of hierarchies to assure efficiency, Napster and NICE can also serve as warning examples for the risks of SPoFs that can result from hierarchies. Networks like native Gnutella, on the other hand, may suffer from signaling overhead due to flooding in a (fixed) heterarchical connectivity structure. Most systems have an invariant structure, they are at most able to change the distribution of data.

16.3 Towards More Self-Organization in Peer-to-Peer Overlays

16.3.1 Active Virtual Peers

Self-organization is seen as an attractive feature of Peer-to-Peer networks as it essentially enables running a complex system without exercising stricter form of control and management. We have seen in Chapter 15 how characteristics and criteria can be defined as a basis for an analysis whether Peer-to-Peer systems exhibit crucial properties of self-organization. In fact, most current Peer-to-Peer systems show less resemblance to self-organization as might have been anticipated. A corresponding analysis has been presented in this chapter. Either criteria of self-organization do not apply at all or they apply only if the Peer-to-Peer system at hand had been configured manually. A truly self-organizing Peer-to-Peer system that exhibits important criteria of self-organization and is also operating autonomously is hard to find.

Peer-to-Peer systems are often referred to as being self-organizing where a coherent behavior emerges spontaneously without external coercion or control. Pure Peer-to-Peer architectures, such as early the Gnutella service, however turned out to be non-scalable. As a response, the need to introduce structure and limited control has been recognized, cf. [398]. In order to introduce heterogeneity into unstructured, pure Peer-to-Peer services, various mechanisms have been proposed. The suggestions range from “ultrapeers” and “superpeers”, as in Gnutella [563] and Kazaa [558] respectively, to distributed mediation servers and peer caches as in eDonkey2000 [185]. These approaches comprise only partial solutions to a more complex control problem. In particular, variability in service demand or load patterns can only be dealt with in a limited way. The demand for services may form hot spots which may shift within an overlay from one location to another one over time. Peer-to-Peer applications may therefore require a more flexible and dynamic method of control and management [160]. In particular, different control methods should be in place when and where needed, should be flexibly usable in combination with each other, and should be extensible in an

evolutionary manner. More generally, it is our goal to introduce and to implement control and structure into Peer-to-Peer applications on demand.

The approach is based on the introduction of the concept of virtual nodes, called Active Virtual Peers (AVP). The proposed approach based on AVPs includes for example a *dynamic* forming and maintaining of Peer-to-Peer overlays or an adaptive routing of signaling and download traffic.

16.3.2 Objectives and Requirements on Control for Peer-to-Peer Overlays

We believe that there exist four areas where the enforcement of control will be beneficiary for such applications.

The first is *access control*. Participants of Peer-to-Peer overlays are typically granted access to all resources offered by the peers. These resources are valuable. Thus, the resource provider, either content provider or network provider, need to identify and regulate the admission to the overlay. In particular for Peer-to-Peer file sharing applications, access control should block off Peer-to-Peer applications or enable controlled content sharing.

The second area is *resource management*. The resources of individual peers have to be treated with care, e.g., low-bandwidth connected peers should not be overloaded with download requests and exploited equally. For Peer-to-Peer file sharing applications, for example, content caching capabilities will improve the performance while reducing the stress imposed on the network.

A third area of interest is *overlay load control*. Overlay load control copes with traffic flows inside the overlay. Its goal is to balance the traffic and load in order to maintain sufficient throughput inside the overlay while also protecting other network services by mapping this load in an optimum way onto the underlying network infrastructure.

Finally, the forth area of command is adaptive *topology control*. Overlay connections may be established or destroyed arbitrarily by the peers since they can join or leave the virtual network at any time. Topology control may enforce redundant connections, thus increasing the reliability of the service. In addition, topology control may force the structure of the virtual network to be more efficient and faster in locating resources when using broadcast protocols.

Having identified the objectives of control for a Peer-to-Peer overlay, it is important to examine how adaptive and un-supervised control mechanisms need to be implemented, without diminishing the virtues of the Peer-to-Peer model or introducing further complexity and overhead to the network. We believe that it is vital to preserve the autonomy of the peers inside a Peer-to-Peer network. Additional control loops, which adapt to the behavior of a Peer-to-Peer overlay, must not interfere with the autonomous nature of any Peer-to-Peer application. To achieve this goal, we suggest implementing control through an additional *support infrastructure*.

16.3.3 An Implementation of the AVP Concept

The main element of the support infrastructure suggested in this section is the Active Virtual Peer (AVP). As its name implies, an AVP is a virtual entity which interacts with other peers inside a Peer-to-Peer network. An AVP is a representative of a community of peers. Its purpose is to enhance, control and make the Peer-to-Peer relation more efficient inside that community. AVPs enable flexibility and adaptivity by the use of self-organization. An AVP consists of various distributed and coordinated components that facilitate different forms of control. By combining these components based on network conditions or administrative policies, we can create AVPs of different functionality.

The AVP performs certain functions, not expected by an ordinary peer. These AVP functions are arranged in horizontal layers as well as in vertical planes, see Figure 16.1. The horizontal layers correspond to the layers on which an AVP imposes control. The vertical separation describes the functional planes of AVPs. These architectural planes have been examined in detail in [160].

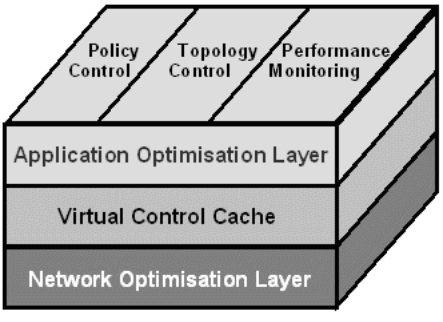

Fig. 16.1: The AVP architectural layers.

The upper horizontal layer of an AVP is called the "Application Optimization Layer (AOL)". It controls and optimizes the Peer-to-Peer relation on the application level. The AOL may apply application-specific routing in conjunction with *access policies*. The routing performed by the AOL is based on metrics such as the state of the peers ("virtual peer state") or the state of the links between peers ("virtual overlay link state") thus changing the peer load and overlay link characteristics such as packet drop rate, throughput, or delay. In addition, the AOL allows for active overlay topology control, which is accomplished in two ways. The Active Virtual Peer may initiate, accept or terminate overlay connections based on access restriction or topology features. Topology characteristics such as the number of overlay

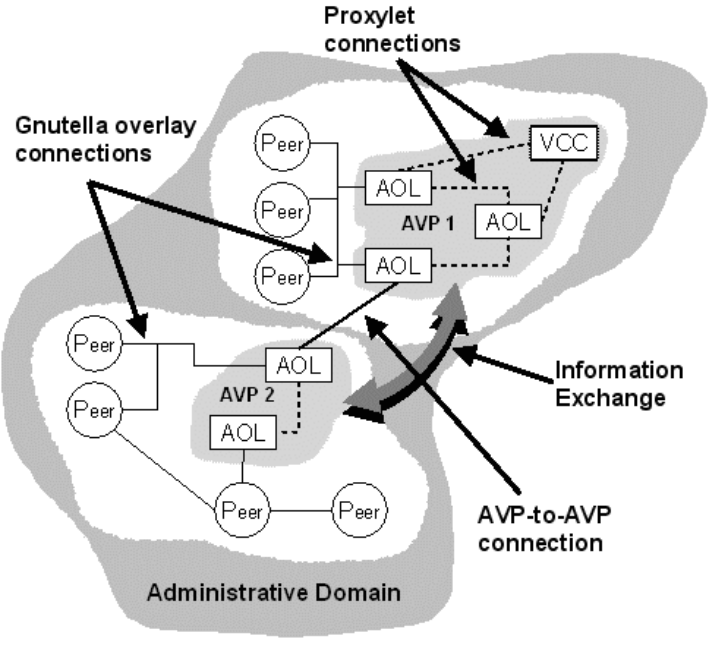

Fig. 16.2: The AVP realm.

connections or characteristic path length can be enforced or may govern the overlay structure. Furthermore, the AOL layer makes also use of the Application Layer Active Networking control mechanisms [242], examined below, for implementing its self-organization features. The AOL can instantiate modules implementing AOL functions whenever and wherever needed. These features enable the AOL to adapt the virtual overlay structure to varying demand, traffic patterns and connectivity requirements by launching new overlay connections and new virtual peers. These self-organization features make the AOL a very flexible architecture.

The middle layer of the AVP is denoted as the "Virtual Control Cache (VCC)". The VCC provides content caching on the application-level similar to conventional proxies. By maintaining often-requested content in close proximity, for instance inside an ISP's domain, large economies in resources and performance gains can be achieved. In addition, the VCC may offer control flow aggregation functions.

The lower layer of AVPs is denoted as the "Network Optimization Layer (NOL)". Its main task is the implementation of dynamic traffic engineering capabilities that map the Peer-to-Peer traffic onto the network layer in an optimized way. The mapping is performed with respect to the performance control capabilities of the applied transport technology. The AVP architecture

may apply traffic engineering for standard IP routing protocols [212] as well as for explicit QoS enabled mechanisms like MPLS [630].

Figure 16.2 depicts a scenario where two AVPs, AVP 1 and AVP 2, are located within a single administrative domain. AVP 1 consists of three AOL modules and one VCC component, while AVP 2 comprises of two AOL modules. Multiple ordinary peers, denoted by "Peer", maintain connections to them. The two AVPs maintain overlay connections to each other. The AOL modules of the AVPs are in command of the overlay connections. This way, the AVPs can impose control on the overlay connection.

Having identified earlier the objectives for control of a Peer-to-Peer overlay, it is time to see how the AVP facilitates these control issues. Deployed AVPs create a realm wherein they constantly exchange information. Each AVP consists of multiple AOL and VCC proxylets which communicate and collaborate. The exchange of information allows for coordinated control of the overlay. A realm of AVPs is more suitable to evaluate the conditions inside a particular part of a Peer-to-Peer overlay than a single entity and this knowledge is distributed in order to achieve better results. Again, this capability promotes the flexibility and adaptivity of the AVP approach. Continuing, an AVP imposes control by providing effectors on connection level. The effectors comprise so far the *Router module* and the *Connection Manager module*. The Connection Manager enforces control by manipulating the connections peers maintain with each other. That is a significant difference compared to most Peer-to-Peer applications where the way peers connect to each other is random. By applying connection management, the AVP can enforce different control schemes.

The *Router module* governs the relaying of messages on application-level according to local or federated constraints, e.g., access restriction or virtual peer state information. The *Sensor module* provides state information for the distributed and collaborative control scheme.

The proposed concept relies on Active Virtual Peers as the main building block. The presented AVPs implement means for overlay control with respect to access, routing, topology forming, and application layer resource management. The AVP concept not only allows for a flexible combination of algorithms and techniques but enables operation over an adaptive and self-organizing virtual infrastructure. The significance of the approach is based on the automatic expendability and adaptivity of the whole overlay network as Peer-to-Peer services evolve. From this perspective, AVPs are inherently different to all other Peer-to-Peer systems. While some other Peer-to-Peer systems do comply with criteria of self-organization, a similar autonomy in developing features of self-organization seems unique to AVPs. AVPs adapt to the environment without manual triggers as opposed to other Peer-to-Peer approaches.

16.3.4 Related Work

The concept of AVPs is similar to the "ultrapeers" since both apply a peer hierarchy and reduce signaling traffic. AVPs differ from "ultrapeers", however, because of their overlay load control capability and adaptivity to the underlying network structure. The well-known Kazaa Peer-to-Peer filesharing service [558] applies a concept similar to "ultrapeers". In Kazaa these distinct nodes are denoted as "superpeers".

The OverQoS architecture [580] aims to provide QoS services for overlay networks. Dedicated OverQoS routers are placed at fixed points inside an ISP's (Internet Service Provider) network and connected through overlay links. The aggregation of flows into controlled flows of an overlay enables this architecture to adapt to varying capacities of the IP network and ensure a statistical guarantee to loss rates. This OverQoS approach complements and extends the limited load control provided so far in the AOL proxylet. However, it lacks any adaptivity to the varying network topology as addressed by the AVP.

Resilient overlay networks (RONs) [26] provide considerable control and choice on end hosts and applications on how data can be transmitted, with the aim of improving end-to-end reliability and performance. However, RONs are mostly restricted within single administrative domains.

16.4 Conclusions

Since self-organization in relation to Peer-to-Peer systems seems often to be identified with even more elusive properties such as emergence, criticality or autonomy, it is difficult to define hard criteria for the existence of self-organization and to apply the criteria rigorously. In a very puristic sense, self-organization in Peer-to-Peer systems can yet only be identified in a very limited form. The question, however, remains whether more self-organization would be desirable. And there is good reason to assume that this would indeed be the case.

Internal feedback is an efficient way to keep a system in balance; the same is true for the state of criticality. *b*-trees are an example from computer science that illustrates the benefit of a balanced system: efficient operational modus, controlled management cost and the existence of upper bounds for the traversal. Such balancing operations don't seem to have attracted yet too much attention in the context of Peer-to-Peer networks. Randomness is often used as a means to circumvent complex operations (e.g., in approximations). A flexible structure is the basis for adaptivity and robustness. A dynamical clustering or hierarchy could be basis for a promising approach. Equality of peers counteract the danger of SPoFs and is widely realized in approaches based on DHTs. A self-organizing Peer-to-Peer system should ideally be able to reproduce its structure, and not only its data. The reproduction should

occur automatically and not be manually triggered or come from the outside. Viable structures should emerge autonomously. Furthermore, the boundaries of a self-organizing Peer-to-Peer network should be self-determined, a property hardly observed with current Peer-to-Peer systems. Detrimental effects caused by attackers or freeriders should be confined or prevented.

Self-organization reaches beyond the obviously desirable properties like flexibility, adaptivity or robustness. It includes the use of random components that allow the system to create new viable structures. Appearances of emergent properties are entailed that are triggered by interacting components. It extends to the state of criticality that allows for appropriate reaction and restructuring to perturbations. A reduction of complexity for a scalable growth is also often seen as a property of self-organization while maintaining identity. In all cases, control that can be exercised externally is limited to a minimum.

A decentralized self-management comes very close to the ideal of self-organization. In addition, it would be desirable if self-organizing Peer-to-Peer networks could be steered towards a certain overall purposeful goal. Such a steering may take place in accordance to observations made with emergent properties of self-organizing systems. Emergence is relying on simple rules, adapts sensibly to perturbations according to an "implanted" goal and does not develop pathological features under various forms of stress but compensates for stress to a wider extend in a reasonable way. Studying emergence may lead to identifying the simple rules to be implanted into Peer-to-Peer networks such that purposeful behavior may emerge. Pathological behavior in terms of detrimental performance or security should be made autonomously avoidable. If self-organizing systems are seen as systems which "create their own life" purposeful and efficient operation may become a big challenge. Self-organizing Peer-to-Peer systems with the inert property of well-behavedness that can be purposefully "implanted" would be the ideal case. How such an "implant" can be created and inserted is one of the big remaining challenges.