Anita Sobe, Dipl. Ing.

Self-Organizing Multimedia Delivery

Towards Emerging Delivery Paradigms for Non-Sequential Media Access

DISSERTATION

zur Erlangung des akademischen Grades
Doktorin
der Technischen Wissenschaften

Alpen-Adria Universität Klagenfurt
Fakultät für Technische Wissenschaften

1. Begutachter:	Prof. Dipl.-Ing. Dr. Laszlo Böszörmenyi
Institut:	Institut für Informationstechnologie
2. Begutachter:	Prof. Pascal Felber, Ph.D.
Institut:	Institut d'informatique, Université de Neuchâtel, Switzerland

Dezember 2011

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende wissenschaftliche Arbeit selbständig angefertigt und die mit ihr unmittelbar verbundenen Tätigkeiten selbst erbracht habe. Ich erkläre weiters, dass ich keine anderen als die angegebenen Hilfsmittel benutzt habe. Alle ausgedruckten, ungedruckten oder dem Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen und Konzepte sind gemäß den Regeln für wissenschaftliche Arbeiten zitiert und durch Fußnoten bzw. durch andere genaue Quellenangaben gekennzeichnet.
Die während des Arbeitsvorganges gewährte Unterstützung einschließlich signifikanter Betreuungshinweise ist vollständig angegeben.
Die wissenschaftliche Arbeit ist noch keiner anderen Prüfungsbehörde vorgelegt worden. Diese Arbeit wurde in gedruckter und elektronischer Form abgegeben. Ich bestätige, dass der Inhalt der digitalen Version vollständig mit dem der gedruckten Version übereinstimmt.

Declaration of Honour

I hereby confirm on my honour that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.
The support provided during the work, including significant assistance from my supervisor has been indicated in full.
The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirm that the content of the digital version is completely identical to that of the printed version.
I am aware that a false declaration will have legal consequences.

Signature:

_____

Klagenfurt, 18. Dezember 2011

Contents

List of Tables

List of Figures

Acknowledgements

I have to exceptionally thank my advisor Professor Laszlo Böszörmenyi for his support during this thesis. I appreciated that he took the time for weekly meetings to discuss new ideas and to evaluate constructively the work so far. He has the skill to make someone look at a problem from perspectives, which are not obvious from the first sight. It was a pleasure to learn from such an exceptional person.

I have to thank Dr. Wilfried Elmenreich a lot who stirred up my enthusiasm for self-organizing systems/complex networks (and Project Euler). I am very grateful for the lively discussions and support, which helped to combine self-organization with multimedia delivery.

My special thanks go to Professor Pascal Felber, who accepted the request to review my thesis and for his valuable input, also during my stay at Neuchâtel in 2008.

This dissertation would not have been possible without the team of the SOMA project (SOMA is a cooperative project of the Lakeside Labs GmbH and the Alpen-Adria Universität Klagenfurt): Dipl.-Ing. Manfred del Fabro, Dipl.-Ing. Marian Kogler, Dipl.-Ing. Stefan Wieser, Dipl.-Ing. Felix Pletzer, Dr. Roland Tusch, Dr. Mathias Lux, Professor Bernhard Rinner and Professor Laszlo Böszörmenyi. I appreciated the collaboration and the many discussions during this project and the great work for the use cases. Special thanks go to Manfred del Fabro for being premium tester of ViNo, discussion partner and coffee kitchen colleague.

I want to thank my family, especially my mother, for their help and motivation during this time and to keep my options open to concentrate on this thesis.

Most importantly I want to thank Dipl.-Ing. Bernhard Dieber for his support, time, humor and patience.

Abstract

In this thesis the non-sequential delivery of media in dynamic networks is investigated. Consider a scenario where people participate at a social event. With the increased popularity of smart phones and tablet computers people produce more and more multimedia content. They share their content and consume it on popular web platforms. The production and the consumption of such media are, however, different from the typical sequential movie pattern: we call this *non-sequential media access*. If the infrastructure is not available, visitors cannot share their content with other visitors during the event. The idea is to connect the devices directly, which is further robust even if people move during the event (*dynamic networks*). Non-sequential media access in combination with dynamic networks brings new challenges for the whole multimedia life cycle. A formalism called *Video Notation* helps to define the single parts of the life-cycle with a simple and short notation. New measures for transport are needed as well. A *caching* technique is introduced that allows for evaluating the goodness of content for being cached based on its popularity in different user groups. However, this cache does not cope with the dynamic network requirement, because such a delivery has to be robust, adaptive and scalable. Therefore, we concentrate on *self-organizing algorithms* that provide these characteristics. In this thesis the implemented algorithm is inspired by the endocrine system of higher mammals. A client can express its demands by creating hormones that will be released to the network. The corresponding resources are attracted by this hormone and travel towards a higher hormone concentration. This leads to a placement of content near to the users. Furthermore, the robustness and service quality is increased by placing replicas of the traveling content along the transport path. Unused replicas are automatically removed from the nodes, to ensure storage balancing. Finally, we show with a use case that a *middleware* based on the hormone-based delivery including well-defined interfaces to the user and to the network can be used for content delivery other than multimedia. For such general application recommendations on possible configurations are made.

1. Introduction

This chapter contains the motivation of the thesis topic. Based on that the contributions are summarized and finally the organization of this thesis is described.

1.1. Motivation

With the increasing popularity of smart phones and tablet computers user generated multimedia content can be created and consumed almost everywhere. Web platforms such as Flickr[1], YouTube[2] and Facebook[3] allow the users to share their *moments* with friends or the public. At live sports events such as the Ironman[4] (a triathlon), visitors produce masses of multimedia data, but there is no specific possibility to exploit this data for live sharing with other visitors. Although organizers provide video walls, visitors cannot influence the content presented. The possibility of querying content such as *"I want an overview of interesting parts of the last 30 minutes"* is missing. Visitors should be able to create their individual presentations depending on their interests. This includes alternative presentation patterns to the typical sequential consumption pattern of traditional movies. As an example an overview of the highlights could be presented as a combination of short videos as split-screen (in parallel).

Such a scenario is dynamic and complex, since visitors move, have heterogeneous devices and have a different notion of *interestingness*. Furthermore, the content quality is different from device to device, the area of the sports event is large, popularity of content changes quickly, etc.

Thus, there is a large potential of research questions in this context. Some of them are: (1) How can a user interact with others? (2) How can videos and images be automatically and/or interactively composed to a presentation? (3) How are the user

[1] www.flickr.com
[2] www.youtube.com
[3] www.facebook.com
[4] www.ironman.com

devices connected to a network? (4) Given a network topology how can content be efficiently delivered to the right places?

These research questions are investigated in the context of the research project SOMA (Self-Organizing Multimedia Architecture) described in more detail in Chapter 2.

The Ironman scenario further shows that content gets more and more important both on the consumer and on the producer side. In this scenario visitors produce multimedia content all around the area, and should also be able to consume anything (multimedia content), anytime and anywhere they want. However, the transport of content is still limited to traditional, mostly static, delivery methods. Future Internet discussions such as described by Hausheer et al. in [9] show that flexible solutions for delivery are needed.

Flexibility means more exactly that the following system characteristics apply.

1. The system is *dynamic* – users join and leave, change their interests, ...
2. The system is *non-deterministic* – it is non-predictable what content will be popular, what the join and leave rate will be,...
3. The system needs *local decisions* emerging to a global state – what content should be moved where to provide better service, ...
4. The system needs *adaptability* for being robust – join and leave rates of users should not harm the rest of the network, alternative delivery paths and content presentations are necessary

These are characteristics of self-organizing systems as defined in [10]. Therefore, I am specifically interested in self-organizing delivery concepts. Since the dynamics of the described scenario take on a key role, predefined structures are not maintainable. Therefore, unstructured peer-to-peer overlays are chosen as a basis.

To realize the delivery, an interface is needed between user requirements and delivery system. From a user's perspective different multimedia files created at a certain event like the Ironman are dependent on each other. As an example consider a visitor who wants to see the best athletes of the swim challenge. To fulfill this request a number of multimedia files might be needed in a specific order. So, a user should be able to

express his/her needs in such a manner that multiple files are addressed.

In traditional systems files are independent. Thus, one research question is to find out how the multimedia dependencies can be implemented. Since these dependencies can vary from user to user, a global index is not manageable. However, a local index on each peer needs sophisticated search algorithms. Additionally, when delivering multimedia content, further requirements apply on transmission quality. A search mechanism has to take care of the transmission of dependent content to fulfill QoS requirements, like low delay, low delay jitter, etc.

The transmission itself can be exploited for influencing the performance of future requests. If everyone only downloads the parts requested, the content cannot spread efficiently to fulfill future requests. By acting as intermediate peer for transport, i.e., being non-selfish, efficient replication decisions can be done leading to an efficient content placement before it is requested. Replicas result in a smaller search space, which in turn leads to higher scalability and robustness.

1.2. Contributions

Self-organizing mechanisms are investigated to cover the process from request until the reception of the required content. In this context a number of contributions are made in five different categories and are described in the following. The corresponding publications are listed in Appendix 9.

1. ***Describe video networks.*** The QoS requirements of users can be expressed by a number of QoS-languages, however, there is no formalism to describe the transport in order to compare different delivery solutions. In this thesis a simple notation called ViNo (Video Notation) is introduced with the goal to match delivery and requirements in one language. Furthermore, the notation allows for the expression of low level QoS requirements and high level metadata requirements. Clients can express their needs for a presentation and a system designer can describe the way of content from the content provider until the end of presentation. Due to the simple notation system decisions can be made and the effort for implementing simulations can be avoided.

2. ***Handle non-sequential media access.*** Media that is consumed more flexible than a traditional sequence has also to be handled differently. On the

example of a simple proxy cache it is shown, that new consumption patterns evolve over time, and such a dependency can be exploited for prefetching. If different users with different taste have to be served, a compromise about the content to be cached has to be made. So, a metric is introduced to evaluate the *interestingness* of content for all users with different taste.

3. ***Bring content near to the user.*** In unstructured peer-to-peer networks the procedure for delivery is usually searching content and if found, downloading it directly from the peer holding the content. Traditional search algorithms, such as adapted Breadth First Search in the Gnutella protocol [11], are QoS-unaware. Either the content is not found or the client is not capable of selecting the node for download according to QoS. One of my hypotheses is that QoS based search (based on, e.g., delay, link load, etc.) can positively guard the later transmission. To achieve this I introduce a self-organizing hormone-based algorithm for searching and QoS-based routing of content. The hormone-based algorithm is managed without global knowledge and consists of two parts: (1) Requests for content are represented by hormones. Hormones are created on an arriving request, are diffused to neighbors and evaporate over time; (2) hormones trigger the transport of corresponding content. The creation of hormones is also influenced by QoS requirements, i.e., for content near to the playback time the hormone concentration is increased. The concentration of hormones diffused to the neighbors is QoS dependent. The better QoS the neighbor can provide, the more hormones are diffused to it. This policy does not necessarily route units on the shortest path, but on the path with the best QoS. Content is placed where it is needed, whereas content that is never requested does not move.

4. ***Reduce search space and increase robustness.*** Typically, search in unstructured peer-to-peer networks does not scale well. I show that non-selfishness of peers (by the provision of some storage space) supports the overall system's performance. By creating replicas on the transport path no further system information is needed. Uninformed and locally informed replication mechanisms are compared with the goal to reduce search space and place content on peers before it is requested.

   The global replica landscape gives an overview of the currently popular multimedia content. Since the popularity of content is assumed to be dynamic, a periodical replication clean-up is suggested and investigated. The investigation targets the comparison of existing clean-up strategies with newly introduced

strategies to underpin their strengths and weaknesses. Furthermore, the replication and clean-up strategies are discussed in a broader context and I show that the robustness of a system can be increased by the combination of local knowledge, replication and efficient storage usage.

5. ***Combine the findings into a middleware for content delivery.*** SOMA describes an architecture that covers the whole multimedia life cycle. The middleware is a generalization of the SOMA distribution layer providing an interface to the user, the delivery and the infrastructure. The hormone-based algorithm transparently delivers and places the content. On the one hand it gets input from the application on what to deliver and on the other hand it gets network related information, such as connected neighbors and low level QoS data of the neighbors. The generalization of the interfaces and the combination with the hormone-based algorithm to a middleware helps to deploy the SOMA distribution layer to a real scenario. Furthermore, the middleware should be applicable in delivery systems for continuous and non-continuous media. Content, other than continuous media might require different parameter settings, e.g., different chunk sizes. Recommendations are made based on a case study.

1.3. Outline

This thesis is structured as follows. This work is done in the context of a project called *Self-organizing Multimedia Architecture*, which will be described informally in Chapter 2. Since the main topics are peer-to-peer and self-organization, Chapter 3 provides background information. The contributions of this thesis are broad, therefore, each chapter contains its own section dedicated to related work. Chapter 4 introduces a multi-purpose multimedia formalism, which is henceforth used for expressing user requests, presentations and QoS requirements and to calculate multimedia transport. In Chapter 5 caching policies are discussed for the efficient usage for non-sequential media. These caching mechanisms are not yet applicable for the usage for self-organizing delivery. Chapter 6 contains the main part regarding self-organizing multimedia delivery, where a bio-inspired algorithm is introduced and combined with replication (the adaptation of the caching algorithm from Chapter 5) and measures for increasing the storage efficiency. In Chapter 7 an artificial hormone-based middleware is introduced, that generalizes the before mentioned characteristics for content delivery. Chapter 8 concludes this thesis and outlines future work.

2. The Context: Self-Organizing Multimedia Architecture - SOMA

This work is part of the SOMA (Self-organizing Multimedia Architecture) project [1].

SOMA strives to cover the whole life cycle of multimedia with a focus on dynamic environments. Dynamic environments can be seen as multimedia systems that depend on the interaction of persons with each other and their devices. The system has to adapt to handle dynamic production and consumption of multimedia content. Examples for such systems are specific traffic surveillance systems and social events, e.g., the Ironman (see Introduction).

If the participants of such a dynamic environment are able to share their most interesting multimedia content with other participants, a *self-organizing human community* evolves. Our goal is to enable participants to see situations through the eyes of the other participants.

At social events visitors are likely to produce videos that have a playback duration of a few seconds up to a few minutes, i.e., small pieces of video. We call such pieces *units*. These units are usually distributed over masses of consumer devices. By making them available to other visitors, new usage patterns can emerge. The collection of such units enables users to *compose* units to an arbitrary presentation. The users can individually decide what to see, in which order and when they want to watch the presentation. This flexibility brings new challenges to all parts of a system.

SOMA defines three parts for the life cycle of multimedia: User, Sensors and Distribution [3]. Users have two roles, consumption and production. Sensors support the users by producing further content and detecting interesting situations. The produced content is then spread to the right places and prepared for presentation.

[1] In this chapter the project is described informally, for a more detailed overview see [3] and a list of publications can be found at http://soma.lakeside-labs.com/

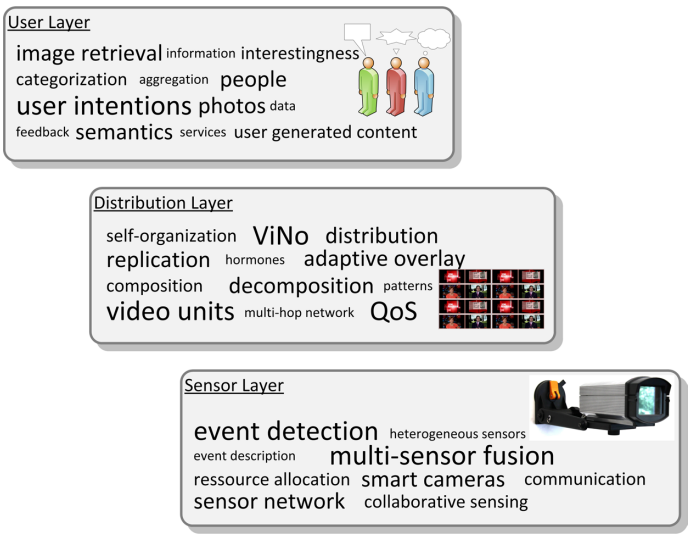


Figure 2.1.: Informal description of the SOMA layers

The three parts form three *layers* and their basic functionality is depicted in Figure 2.1. The figure subsumes all topics of interest for each layer in a tag cloud. Larger tags represent focus topics, which are further described in the following:

1. **Sensor layer.** Sensors support users in the production of content by automatically detecting interesting events. This is done by the aggregation of low level sensor features to higher level events. What a higher level event actually is, is decided locally by the sensors (see [12]).

2. **Distribution layer.** The distribution layer consists of three sub-layers.

   - Composition and decomposition: All content produced by the users is going through this layer to detect semantically meaningful units. If the content gets too large, decomposition is necessary.
     Users are actively integrated in the process of "making their own movie". Since this sub layer represents the interface to the client, it helps to express compositions interactively. Besides that, also automatic compositions are possible, e.g., for video walls (see [5], [13]).
   - Resource management and actual delivery: This layer bridges the gap between the actual network and the user by exploiting QoS information provided by the next sub layer.
   - Infrastructure management: Visitors can participate with their mobile phones and tablets, as well as by using computers provided by the infrastructure of the event. All devices build a random, unstructured overlay,

or can be clustered to so called Flocks [14]. Flocks are clusters relying on a dynamically built system wide QoS-map.

3. **User layer.** The user layer uses implicit and explicit feedback from the consumers as well as from the producers and aggregates this information to enrich the content with additional metadata. This information has not only impact on future presentations, but also on the delivery of units (see [15]).

2.1. Interfaces to the SOMA Layers

This thesis covers the resource management and delivery sub layer of the distribution layer and is therefore connected to the composition/decomposition sub layer and the infrastructure sub layer.

In Figure 2.2 the relation to the decomposition/composition sub layer is described. The decomposition layer provides atomic units, containing one semantic part of a video or a photo and all extracted metadata. The interface to the composition layer is defined by a formalism called Video Notation (ViNo) that allows for the expression of requests. ViNo further allows for QoS expressions, also defined by the composition layer. ViNo is part of this thesis and is described in Chapter 4.

The infrastructure layer is responsible for connecting the devices of the visitors and to build a dynamic overlay. Thus, this work relies on an existing overlay and the notion of nodes and neighbors.

2.2. Use Cases

In 2010 we performed three use cases to collect user material and to analyze typical behavior of visitors at larger events. The use cases showed advantages and disadvantages of traditional as well as of dynamic systems.

2.2.1. SOMA World Games

The aim of the SOMA World Games event was to integrate all layers and their so far available capabilities.
We provided four interactive video games (e.g., Karaoke, interactive sports, ...) and simulated that each of the video games can be played in one of four cities around

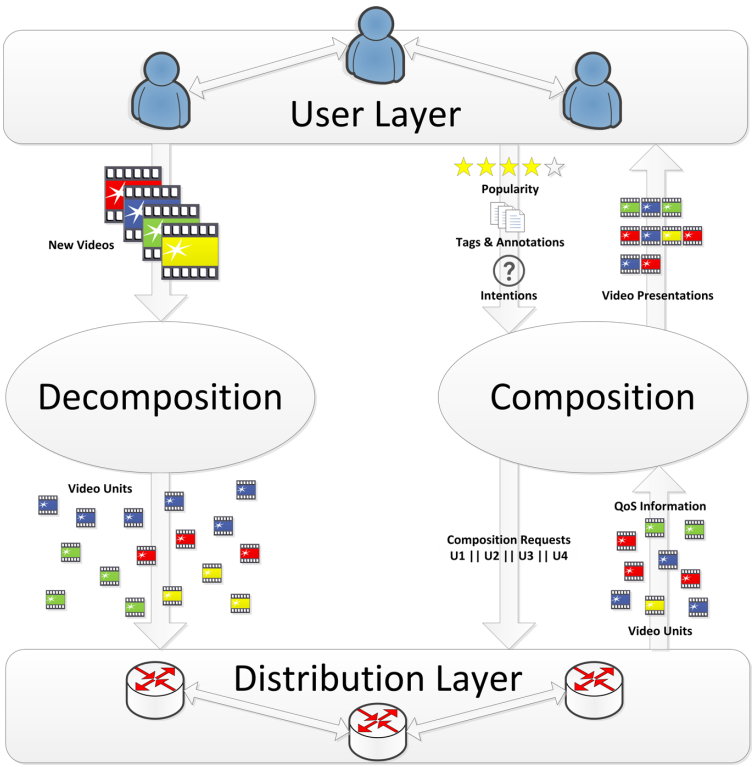


Figure 2.2.: Overview of the composition/decomposition/distribution relationship [3]

the world, namely New York, Munich, Tokyo and Rio de Janeiro. Visitors at these games were able to play and to take photos and videos. Furthermore, they were able to see automatic compositions from interesting scenes of the own and the other cities.

In this use case we integrated the sensor and the distribution layer and created a data set for the user layer for further analysis. At the sensor layer simple events were detected, such as *many people* and *not many people*. After that the most interesting parts were selected and sent to the network. The network was emulated by using PlanetLab[2], a distributed test bed usable for research purposes. The content of the virtual cities had been injected into the network near the real cities. Thus, the content had to travel through the real network before it could be presented.

A simplified global index was created to allow the composition sub layer for creating automatic presentations. Simple patterns such as *Munich*, *Highlights*, etc. were the basis for the presentation.

[2]www.planet-lab.org

With this event we were able to define further requirements for our proposed algorithms. Furthermore, the acceptance of such a system for visitors was validated.

2.2.2. Anniversary Festival at the Alpen-Adria University Klagenfurt - UniSommer

This event was targeted at past, current and future students and at researchers from the immediate region. Initially planned as presentation platform for researchers the event evolved to a static program of official presenters and an exhibition displaying the history of the university.

At this festival we evaluated the applicability of our concepts to a rather rigid environment. In contrast to the SOMA World Games, the focus was set on the composition sub layer. Since this event was larger than the first use case we were able to identify producer patterns. We presented the content to the users by automatic compositions on different screens and at a number of terminals, where users were able to interact with the system. The interaction was also used as a feedback loop that influenced further presentations. The presentation of user generated content was very popular and it showed that a system like SOMA would be accepted by a broad audience.

2.2.3. The Long Night of Research - Die lange Nacht der Forschung

The aim of the final use case was to present our work to a larger public. The Long Night of Research attracted more than 5,000 visitors to the presentation of 105 research projects over a few hours. The area of the presentation covered the university itself and the greater campus. Thus, this scenario is perfectly matching the target scenarios of SOMA. The influence of the visitors on the presentations was further emphasized and stations with simple software for uploading and consuming videos were deployed on different places on the campus.

The goal of this use case was to get a data set of all media provided by the visitors including metadata and to collect information about consumption and production behavior of the visitors.

We decided for a simple deployment where all data was uploaded to one server, where it was further prepared for the presentation. The decision for the central server

was made to ensure controllability. However, some of the presentations were highly popular and encouraged even more people to participate with their own content – as a consequence flash crowds lead to slow service of the server which resulted in recurring presentations of content. The use case event showed that controllability has its costs and that a dynamic delivery system would have adapted to such issues, but would have been harder to set up.

3. Background

The aim of this thesis is to identify possible delivery methods to manage dynamic situations. Therefore, this chapter covers two topics: traditional delivery over peer-to-peer systems and the basics of self-organization for future delivery methods.

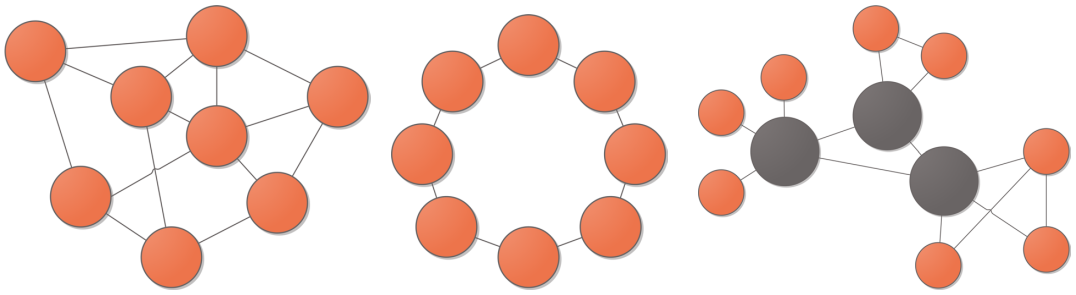
3.1. Content Distribution in Peer-to-Peer Networks

Peer-to-Peer networks are known to be scalable and fault tolerant and their applications are wide spread from communication systems, Internet service support, database systems and content distribution [16]. This section focuses on content distribution.

3.1.1. Basic Concepts

Peer-to-Peer networks consist of nodes of equal roles and have a notion of neighborhood. Since this neighborhood does not necessarily represent the physical network connection, we speak of a logical *overlay* [17]. Such overlays can be either *unstructured*, *structured* or *hybrid* as shown by example in Figure 3.1.

1. **Unstructured networks**.
   Nodes join and leave as they wish, connecting to a number of other nodes. The advantage of an unstructured architecture is the capability of handling the high dynamics of peers leaving and entering the system. However, because of these dynamics it is hard to hold an index of the provided peer content. Another problem is to enter the network. To find other peers already connected, a number of bootstrapping strategies may be applied as described in [18]. (1) One or a number of bootstrapping servers hold lists of currently available peers; (2) The peer sends a broadcast or a multicast to find other peers; (3) peers already in the overlay send periodically broadcast messages containing their connection information; (4) the peer tries to contact a peer in its cache. In addition to these four strategies a number of requirements for a successful connection exist.



(a) unstructured (b) structured (c) hybrid

Figure 3.1.: Examples for peer-to-peer overlay architectures

A peer should connect preferably to another peer, which is close in terms of network proximity and does not have already a high number of other connections.

2. **Structured networks**.
   The content of the nodes is tightly bound to the logical location of the node. Nodes are organized to build a given structure, e.g., a ring. These structures are used to efficiently locate specific content. Overviews and examples of structured networks can be found in [16], [18] and [19]. A specific example is a distributed hash table (DHT). In a DHT each resource and each participating node can be identified by a unique key. The key defines where the corresponding resource will be located, i.e., its similarity to a corresponding node identifier (id) specifies its location. In other words a node is responsible for a range of keys depending on the system's definition of key-similarity. The query routing is done based on the node ids and expects concrete requests, i.e., in most systems range queries are not supported. Furthermore, there is no possibility to take advantage of popular files.

3. **Hybrid networks**.
   A hybrid network consists of two tiers. The first tier comprises a number of so called super-peers, which have a higher capacity than usual peers and are expected to be always online. The super-peers build an overlay. The second tier comprises usual peers that connect to other usual peers and/or to super-peers. The super-node overlay can be organized as a DHT (see [19]), such as seen for trackerless BitTorrent. Traditionally, peer-to-peer systems are regarded as

cheap shared storage without the need of system maintenance. Super-peers that are expected to be always online invalidate these characteristics.

In dynamic systems any structure is hard to manage, therefore we focus on the use of unstructured peer-to-peer networks and describe in the next section existing possibilities of content search.

3.1.2. Search in Unstructured Peer-to-Peer Networks

In structured networks the lookup of content is very efficient, some implementations reach a complexity of $O(logN)$, where N is the number of nodes in the overlay. If the nodes' behavior is very dynamic, i.e., peer churn is high, the management effort for keeping the structures is influencing the search performance. Unstructured networks are aware of the peer churn and the algorithms adapt to the given situation. The drawback of the non-existing structures is that search is less efficient, which is proven for the traditional search methods *flooding* and *random walk*.

Flooding can be compared to Breadth First Search. Each node forwards the query to all its neighbors, where advanced algorithms detect and avoid circles. Gnutella is an example for flooding based search, it uses a time-to-live-limited Breadth First Search [11]. This search method is executed by forwarding a message to all neighbors until a time-to-live threshold (TTL) is reached. All results are routed back along the path the query took. The extensive forwarding of messages is robust, but leads to a high number of messages and is therefore inefficient.

Random walk is a strategy to reduce the number of messages by randomly choosing a neighbor for forwarding the message, but at the risk of not finding the desired content. The most often used random walk adoption is *k-random walk*, where k random searches are started.

To overcome the disadvantages of the two basic techniques researchers proposed a number of search strategies, compared and subsumed in [18], [20], [21] and [22]. In the following a selection of these strategies is presented.

Kalogeraki et al. proposed two methods in [23]. The first method uses Breadth First Search, but limits the number of neighbors for message forwarding. Each node randomly chooses the half of its neighbors and forwards the query message to them.

So, this method can be seen as an adaptive version of k-random walk. The second method, called *intelligent search*, requires each node to store a profile of the last replies of its neighbors. The profile calculates the similarity of queries already fulfilled by this neighbor. The more similar queries were fulfilled the more likely the query will be forwarded to this neighbor.

Another search mechanism called *routing indices* is introduced in [24]. It concentrates on text documents where each of the documents covers a specific topic. If a query for a document is arriving, the path to which the query should be routed is estimated by a *goodness* calculation. This goodness defines the number of related documents in a given direction, but it does not consider the hop distance to the documents. Therefore, the goodness calculation is extended to find the most documents within the shortest range. The new routing indices represent aggregated information until a predefined hop distance.
The quality of this algorithm depends on the topic definition. If the topics are defined too detailed a user may not find the required content. If the topics are defined too coarse the returned content may be too general to meet the user's expectations.

Yang and Garcia-Molina proposed to use *iterative deepening* in [25]. The query source node initiates periodically Breadth First Searches with increasing TTL until the query is satisfied or in the worst case a predefined maximum depth is reached. The authors expect a system wide policy that specifies a list of depths and with it the maximum number of Breadth First Searches. As an example the list {a,b,c} defines three searches with depth a, b and c. If a request with TTL a arrives at a node at depth a, but the node cannot satisfy it, the node stores the request. If a command for Breadth First Search with TTL b arrives it forwards the stored query to a neighbor. This prevents multiple processing of the same query.

A more dynamic approach is the *adaptive probabilistic search* (APS) introduced by Tsoumakos and Roussopoulos in [26]. The search principle is an extension to random walk, and chooses k walkers to search for a specific object. A node can act optimistic or pessimistic. The node increases the success probability of a chosen neighbor assuming the search in this direction will be successful (optimistic). If it decides for the pessimistic behavior, it decreases the probability of a chosen neighbor. Initially, the node is more likely to choose the optimistic strategy. If the estimation was incorrect an update procedure corrects the probabilities along the path back to the requester and the node switches to the pessimistic strategy. Although APS performs

better than random walk, the update procedure requires additional messages. An enhancement chooses to switch from optimistic to pessimistic strategy more efficiently by monitoring the successful queries. If the number of successful queries is high the ratio of optimistic decisions is increased, otherwise pessimistic decisions are preferred.

Lin et al. propose in [27] a search algorithm that generalizes flooding and random walk. The algorithm, called *dynamic search*, differs two phases depending on the current hop count h and the decision threshold n. If the hop count is smaller than n flooding is used, otherwise the policy switches to random walk. This ensures that content is found in any case within a specific search space. The authors evaluate different values for n. If the maximum hop distance is seven, the best search efficiency is reached if $n = 2$. However, the parameter has to be recalculated for each network scenario.
The authors further propose to substitute the random walk algorithm by knowledge-based algorithms, such as APS. The performance further increases because of the good relationship between search performance and cost.

Whereas the before mentioned methods rely on a large overlay, a number of researchers build dynamic sub-overlays to cluster nodes with similar content. An example for this is the *acquaintance mechanism* introduced in [28]. An unstructured network consists of neighbor links, chosen randomly, and acquaintance links, chosen because of common interests. A peer connected via acquaintance link is called friend, of which the peer also has state information. Queries are routed through the network and acquaintance links are dynamically added or adapted based on successful query fulfillment. The resulting overlay clusters nodes with similar interests, but leaves usual (random) links for global connectivity.

3.1.3. Content Delivery Applications

In current content delivery systems the search mechanism is often detached from the actual distribution. The distribution, however, is optimized for fulfilling multiple requests for the same content at the same time (multicast), which is often the case for multimedia content that is either streamed or downloaded. Li in [19] categorizes delivery methods into *tree-based delivery* and *mesh-based delivery*.

Tree-based delivery integrates peers on the intermediate path to realize application layer multicast. Peers interested in a specific file form a tree-like overlay. This

structure leads to a fixed delivery path for the whole transmission. All intermediate peers forward and replicate the content, whereas the leaf nodes do not contribute. If an intermediate peer leaves the network, the delivery tree has to be rebuilt which leads to interrupted service for the subtrees of this node.

Therefore, *SplitStream*[29] splits up the content into pieces and creates for each piece a different tree. For increasing the robustness, *CoopNet*[30] additionally uses multiple description coding (MDC) [31]. With MDC the content is partitioned in substreams (either spatially or temporally), where a substream is referred to as a description. The descriptions are independent, such that any combination of descriptions reaching the target leads to a consumable video. The more descriptions reach the target, the better is the final quality of the video.

Another example for tree-based streaming is *CrossFlux* [32]. CrossFlux generates one tree-based distribution overlay per stripe of a file. In each tree overlay the nodes adapt their placement by applying local rules. Nodes with high bandwidth are placed near to the root and the nodes with low bandwidth are placed near to the leafs. Furthermore, each tree consists of primary links and backup links. If a node fails the backup links help to reconnect the overlay. The authors further used the findings of the distribution trees and analyzed the distribution flows in mesh-based networks [33]. In mesh-based delivery systems nodes connect to a number of peers that hold the requested content. The authors found out that the distribution flows often follow a tree-like structure and therefore optimized the overlay in the same way as for the tree-based overlay, which increased the throughput of the network.

One of the most famous mesh-based content delivery systems is *BitTorrent* [34]. BitTorrent divides files into chunks of fixed size and the chunks can be downloaded from several peers in parallel. Each node has to decide (1) which chunk to download next and (2) from which peer it is downloaded. The basic policy is to download the rarest chunk from the connected peers.

Recent activities consider adaptations of BitTorrent for multimedia streaming. Since the default protocol of BitTorrent is not applicable for streaming [35], other piece picking algorithms are discussed. Vlavianos et al. propose *BiTos* in [36] to balance parallel and sequential downloading of pieces to fulfill the QoS requirements for streaming and to provide efficient delivery. This is done by adapting the piece picking algorithm from rarest first to a probability based algorithm. Pieces are first

categorized into three classes: already downloaded/missed, near to playback and rest. With a given probability pieces are more likely to be downloaded from the near to playback part than from the rest part. Pieces from the rest category can be downloaded in the rarest first manner, while the near to the playback pieces have to be downloaded sequentially.

An example for real-life applications of peer-to-peer streaming is *PPLive* [37], [38], which is a popular IPTV service in Asia. PPLive follows a double buffer strategy, which allows the peers to efficiently exchange content and to cover jitter. Although streaming only supports sequential download of pieces, PPLive only needs some nodes acting as seeders. The bandwidth of these seeders is adaptively allocated depending on the measured health of the channels. Additionally, peers that store the whole content so far also act as bootstrapping servers, which unfortunately leads to high start-up delays, e.g., up to 2 minutes for less popular channels.

Apart from peer-to-peer systems, the latest multimedia streaming trend is *HTTP Streaming* [39]. HTTP streaming targets the traditional Internet, which is based on TCP. TCP is not popular for video streaming, because of its congestion control and slow-start policy. However, HTTP over TCP is traditionally used for progressive download (see YouTube), which allows for watching the content immediately if a significant part is already downloaded. Progressive download, however, is inflexible in comparison to traditional streaming, because users have to preselect the quality version of the content and then stick to that version, which can lead to bad experience if the infrastructure does not support this decision. Furthermore, VCR actions, such as fast forward and seek/play, are limited or not supported.

The advantage of progressive download is its simplicity and therefore researchers are interested in improving it by *adaptive streaming*. This approach assumes that the content is split up into segments. Instead of requesting a whole file, the client requests individual segments via HTTP. This enables VCR actions, because the segments are small and easily accessible. Furthermore, the segments are available in different qualities, therefore the client can also switch between different bitrates with each request for a segment. The different qualities can be encoded in SVC [40], where a base layer defines the minimum quality and a hierarchy of enhancement layers can be used for higher quality transmissions.

3.2. Self-Organization

Self-organization is a hot topic for computer science, because of the increasing *dynamism and complexity* of current systems (see Internet and Web 2.0). It is hard to predict the advances of a system because of circular dependencies [10], [41]. A global controller can therefore hardly be designed. A self-organizing system consists of distributed entities that *interact* with each other [42]. These entities perform *local decisions*, which at some point evolve towards a global equilibrium [10]. The decisions are influenced by *positive and negative feedback*. Positive feedback amplifies the system and leads to a snowballing effect. This can be controlled by the negative feedback. The *global pattern* evolves due to the interplay of positive and negative feedback.

A traditional example for positively and negatively influenced patterns is the human population. In the recent centuries the births led to a higher population, the offsprings reproduce themselves, which again leads to an even higher population. At a specific threshold the population might be regulated, either by a higher death rate or a lower birth rate [42]. The positive and negative feedback allows for adaptability and guarantees *robustness*. If a part of the system fails, the overall system may suffer, but this will not lead to a sudden breakdown. In contrast to a centralized system: If one would remove the processor of a computer, the overall system will be unusable [41].

In the 1940s researchers started to discuss the design of artificial self-organizing systems [43]. From this time on numerous proposals were published for applying principles found in nature to the technical world. These systems can be subsumed under the term *bio-inspired self-organizing systems*. Depending on the problem context the application of biological patterns can be done in two ways—top-down or bottom-up. Either the problem context is found in nature (and the possible solutions are analyzed and rebuilt, e.g., airplane wings), or the problem is abstracted from its natural context (e.g., ant-based approaches for routing) [44].

3.2.1. Applications of Bio-inspired Self-organization

Babaoglu et al. [45], Mamei et al. [46] and Dressler and Akan [47] collected examples for bio-inspired applications for computer science and networking.

For instance researchers abstracted the behavior of ants for routing purposes, epidemic patterns for information spreading, genetic algorithms for pattern formation and firefly flashes for heartbeat synchronization. In the following these applications are discussed in more detail.

Swarm Intelligence and Social Insects

The most popular pattern is Ant Colony Optimization (ACO) as surveyed by Dorigo et al. in [48]. ACO is inspired by some species of ants that indirectly communicate by depositing pheromones in the environment. The ants tend to follow stronger pheromone concentration, which leads them to a near food resource. If no pheromone is found, the ants follow random paths, which might lead them to a food resource. If the ant is the first, it will return earlier than the others and on the way to the nest it spreads pheromones for the other ants.

As described by Dorigo et al. in [48] ACO can be described as a metaheuristic, applicable as a framework for combinatorial optimization problems (e.g., the traveling salesman problem). The ACO metaheuristic consists of three phases: (1) solution construction, (2) improving the solutions by local search and (3) pheromone updates. The construction of solutions is defined by the trail following behavior of an ant. It follows a stochastic mechanism that is implementation dependent. The local search is an optional step to which performs pheromone updates. Pheromone updates emphasize good solutions and reduce the number of bad solutions.

As an example the first implementation of ACO was *Ant System* as described in [49], followed by a prominent improvement called *Ant Colony System* [50].

Ant Colony System is an implementation for solving the traveling salesman problem. The salesman has to find the shortest path visiting a number of given cities, with the restriction to visit every city only once. Analogue to that the Ant Colony System expects as an input a graph of connected cities. The links between the cities can hold pheromones and costs, which are defined as the distance between two cities. Agents represented by ants travel around the graph and spread pheromones. The decision to which neighbor the ant routes further is defined as state transition rule. The ant can follow two strategies: exploitation and exploration. In the first strategy the ant follows the link with the lowest link costs and a large amount of pheromones. The second strategy should ensure that also alternative paths will be tried by selecting a

link randomly.

The traveling salesman problem was also translated to general routing problems in networks, e.g., by *AntNet* [51]. The ants are used to build routing tables in packet-switched networks. Forward ants search for the resource and if found and if the goodness of the path is sufficient, the routing tables of the intermediate nodes have to be updated. To solve this, backward ants are created and travel the way back to the origin. To get the latest information of the network, a node regularly creates forward ants. The number of ants created depends on the current network conditions to avoid congestion. The destination of the forward ants is chosen randomly among all nodes in the network. To which intermediate node a forward ant moves is dependent on a state transition rule that considers link queue lengths. One important part in routing is the prevention of cycles. The forward ant simply stores information about nodes already visited and excludes them from the state transition calculation. If the ant is forced to travel in a cycle, it continues until its lifetime is over.

Another phenomenon of social insects is called *brood sorting*[52] performed by specific species of ants. These ants build their nests in existing crevices in rocks and by sorting their brood the ants optimize the feeding procedure. If food is restricted some brood stages are prioritized for better survival. The ants usually use only one nest chamber, and they arrange their brood in concentric rings to a cluster somewhere away from the entrance. Each ring contains a different brood stage, where similar brood stages are placed near to each other. At nest creation the brood items are placed randomly, however, later the new brood items tend to be placed near to similarly staged brood items. In between a resorting of brood items may occur. If a brood item is detected that is not similar enough to the neighboring items, it is picked up and relocated to a place where it fits better. Additionally, brood items from small clusters are more likely to be picked up and brought to larger clusters that finally build the single concentric cluster of brood items.

This model was applied to distributed sorting problems, e.g., by Melhuish et al. in [53]. The authors rebuilt the principles in homogeneous robots and let them sort different colored plates. The experiment showed that agents can solve sorting problems much simpler and without the need of specific sensors for spatial orientation.

Epidemic Spreading

Epidemic spreading refers to biological models that analyze virus transmissions. In computer science this principle is used for information dissemination in a dynamic network, e.g., for configuration dissemination for sensor networks [47].

In such a system nodes are connected in a network and communicate according to a diffusion algorithm. In an optimal scenario, this algorithm takes care of spreading the information to the whole network. However, the success of epidemic spreading is dependent on the network structure. In a random network an epidemic spread tends to die out quite fast. If the network can be characterized as a scale-free topology such as the Internet and social networks, the epidemic spread is persistent.

Dynamic networks such as mobile ad-hoc networks (MANET) need a mechanism for fast and reliable information dissemination, therefore epidemic spreading is popular [54]. The epidemic spreading mechanism categorizes the participating nodes in susceptible nodes and infective nodes. The task is to transfer susceptive nodes into infective ones. Khelil et al. [54] define susceptive nodes to be interested in specific information and infective nodes as those who carry information. When the users move around with their nodes holding information, the nodes advertise summaries of it. Any other node within the sending range consumes information of interest and further advertises this information. Khelil et al. further defined a mathematical model for the infection rate of a system. The infection rate is dependent on the probability of information transmission, the number of contacts of the nodes and on the number of the overall nodes in the system.

Further examples regard the routing of messages in MANETs, eventually reaching the target [55]. Infective nodes (holding the packet to deliver) infect all other nodes they meet, which in turn infect other nodes. Thus, the population of infective nodes increases. If the destination is infected the delivering node removes the packet from the buffer, to not propagate it further. Additionally, it blocks the reinfection with this packet by storing the state "packet delivered". Adaptations of this basic mechanism regard the overhead reduction of copied packets, e.g., by introducing a probability of infecting a node that is not the destination.

Genetic Algorithms

In the early 1970s John Holland [56] described the idea of adapting evolution and natural genetics for applications in computer science to find optimal solutions in a large search space. The fittest solutions can reproduce themselves and the optimum evolves over several generations. Holland's algorithm is widely defined as *Simple Genetic Algorithm (SGA)* [57].

The basis for the SGA is a population of bit strings, each representing a solution of the search problem. A generation evolves by *selecting* the fittest and let them create offspring. Offspring are copies of their parents that might with a certain probability evolve to new candidates by *crossover* operations or *mutation*. The default crossover operation selects a random pair of strings and parts of both strings are interchanged. E.g., two strings have length l and a crossover point $1 < x < l - 1$ is defined. At the crossover point the strings are cut and the halves interchange and build two new strings. A mutation is implemented by flipping one or more bits of a given string.

Each resulting string is evaluated according to an implementation dependent *fitness function*. The goodness of the string is the result of the comparison of the average fitness of the population and the fitness of the string. The fittest strings are selected by applying the *roulette wheel selection algorithm*, where each string gets a sector of the roulette wheel. The higher the fitness, the larger is the sector. Then a random number is created, which matches one of the sectors. The selection is repeated until the predefined size of the next generation is reached. The procedure of selection, crossover and mutation are performed until a termination criterion is matched, e.g., maximum execution time, maximum number of generations.

The SGA has been adapted in numerous ways, however, a genetic algorithm typically consists of selection, mutation and crossover. Researchers exchanged the before mentioned selection, crossover, and mutation mechanisms by other solutions. For instance the selection mechanism requires a string to win tournaments against a selected set of strings. The best k strings in a k-ary tournament are then taken for the next generation. Furthermore, there are dynamic genetic algorithms that adaptively manage the crossover and mutation probabilities. Finally, distributed and parallel implementations ensure lower computing times.

Firefly Synchronization

In different areas of the world summer nights are characterized by the flashes of fireflies. Different species evolved different patterns of their flashes that guide the flying male fireflies to the right species of females. One type of fireflies in Southeast Asia is specifically interesting for researchers, because these fireflies seem to perfectly synchronize their flashes without global control [42].

Since synchronization in distributed systems is an important topic, models of firefly synchronization were developed. One example is described by Babaoglu et al. in [58]. The authors investigate heartbeat synchronization in peer-to-peer networks by implementing firefly synchronization. Each node is seen as oscillator, which is characterized by a phase and a cycle length variable. The phase increases linearly from 0 to 1 in a number of time units (the cycle length) and fires a flash to its neighbor nodes if 1 is reached. Then the value of the phase is reset to 0 and the cycle starts again. In peer-to-peer systems transmission delays vary and failures have to be considered. Therefore, the proposed synchronization protocol divides the synchronization into submission and reception of flashes. The sender waits until the phase reaches its threshold and then fires. Then, the most important task is done at the receiver. On reception of a flash the threshold of its own phase and/or its own cycle length has to be updated. The authors propose different models for that update. The simplest model only updates the phase according to the received flash, i.e., fires earlier or later. Without failure, this model leads to pairwise synchronization if the cycle lengths of all nodes are identical.

Artificial Immune System

As other self-organizing systems the artificial immune system (AIS) consists of a number of interacting entities. Most AISs are a simplified model of biological immune systems. The human immune system consists of two parts, the innate immune system and the adaptive immune system, which are tightly linked. However, most computer models concentrate only on the adaptive immune system.

The immune system is able to differ between the body and external pathogens (e.g., bacteria, viruses, etc.). With sophisticated pattern matching the response of the immune system to a detected pathogen is created. The response consists of antibodies that connect to invading antigens [59].

The immune system consists of a number of different cells that detect and respond to pathogens. The most important cells are B and T cells. B cells are lymphocytes that are programmed to create specific antibodies, thus are important to detect pathogens. T cells are also lymphocytes, but act as regulators for other cells and can directly attack infected cells [60].

According to Dasgupta et al. in [59] the most discussed models of AIS are *immune network models, clonal selection and negative selection*. The immune network model assumes a number of connected B cells to detect antigens. Initially, a B cell network is taken out of a training set. This network is then exposed to antigens. B cells that successfully bind to the antigens are cloned and eventually mutate for further classification. If the binding was not successful, a new B cell is created with the antigen as a template and integrated into the network. The negative selection algorithm concentrates on the detection of self-cells, i.e., the body. T cells are tested by self-proteins before brought to the body. If the proteins bind, the cell is destroyed, otherwise the T cells are allowed to escape the thymus and enter the body.

3.2.2. Specific Applications for Content Delivery

The examples described before are copied from the behavior found in nature. In this section the focus is set on more specific examples towards self-organizing algorithms for content delivery, such as ants for resource management in grids, ants for content search and routing and artificial immune systems for content delivery.

Service and Resource Management in Grids

In dynamic grid networks a major task is to discover services and resources. For faster lookup, descriptors are disseminated over the grid, i.e., a distributed index is created. This can also be compared to content discovery in peer-to-peer networks.

Forestiero et al. proposed a descriptor sorting and replication algorithm for grids called *Antares* in [61]. This approach is not only ant inspired, but can also be categorized as *brood sorting* such as described in [46].

The descriptors are bit strings encoded by locality sensitive hash algorithms (see [62]). In contrast to standard hash algorithms, similar content results in similar hash values. This helps to sort similar content based on its hash value. The sorting is done by ant-inspired agents that travel the grid and pick and drop descriptors. According

to a given probability an agent picks one or more descriptors. Descriptors that are most different from the rest of the descriptors on the current host are more likely picked. Agents operate in two modes, copy and move. In the copy mode the agent creates a replica of the descriptor, whereas in the move mode not. The transition between these states is pheromone based. In the beginning, each agent starts in the copy mode. At some point, when the resources are better sorted, the activity of the agent decreases. Then, the agent starts to increase its pheromone level until a given threshold, then it switches to the move mode.

An agent decides to drop a descriptor based on the node's *centroid*. The centroid is a measure that aggregates the description of the node's content. If the descriptor carried by the agent matches the centroid of the node, a drop is likely.

It is interesting that Antares has also been adapted to support QoS-based picks and drops [63]. Descriptors with better QoS are more likely to be replicated. The QoS is described as a non-negative real value, where higher values describe higher QoS. Analogous to the pick probability the drop probability increases if a descriptor with high QoS is carried by an agent.

Ant-based Content Search in Peer-to-Peer Networks

Michlmayr proposed in [64] a search algorithm for unstructured peer-to-peer networks. The so called *SemAnt* algorithm extends *AntNet* for query routing. A query is represented by a number of ants. Since in a peer-to-peer system the target node is typically unknown, Michlmayr introduces a TTL parameter for forward ants. A search procedure terminates, if a resource is found or if the TTL is reached.

Michlmayr investigated two implementations of the algorithm. In the first case a search is successful if one instance of the desired content is found. A backward ant is created to tell the requester the location of the content. In the second case the goal of the search is to find as many results as possible. So, after creation of a backward ant, the forward ant travels further. The search is successful if the predefined number of results is returned.

At startup SemAnt can be compared to k-random search, however, backward ants spread pheromones and therefore the next forward ants can be guided to the resources. The limitation of this algorithm is that the content location is assumed to

be static. If the peer with the resource leaves the system, the pheromones guide the forward ants into the wrong direction until they evaporate.

Another example for search in peer-to-peer networks is called *AntSearch* [65]. This approach is specialized in identifying free-riders. The pheromones are not mapped to content as done by SemAnt, but they are used for indicating the reliability of a peer. If a peer has a low pheromone value, it is avoided to forward messages to it. Thus, the number of unnecessary messages is reduced. The local pheromone value is stored in every peer of the system and consists of two parts. The first part represents the number of hit queries and the second part the number of processed queries. The evaluation of the goodness of a peer is based on the local pheromone value, but depends also on the average pheromone value of the neighboring peers.

The search algorithm consists of two phases, a probe phase and a flooding phase. The probe phase is a short range flooding to estimate the popularity of the requested content. The popularity of the requested content allows for adaptive picking of the flooding parameters. For example, popular content is very likely to be found in the near neighborhood, therefore the TTL parameter can be small. Furthermore, peers only forward to those neighbors having a high pheromone level.

Artificial Immune System Inspired Replication

Ganguly et al. proposed in [66] a search algorithm that implements an artificial immune system. A node initiating a query generates an antibody for it. Files are replicated over the network and represent the corresponding antigens. The antibody can be seen as message traveling around the network finding corresponding antigens. The matching is done by calculating the Hamming distance between the antigens stored on a node and the traveling antibody. If a match was found the query origin is contacted, otherwise the antibody proliferates. The proliferation can be done in two different ways, the simplest proliferation mechanism creates a number of replicas and spreads them to randomly chosen neighbors. The random neighbor selection does not consider nodes that are already visited. Therefore, the proliferation mechanism is extended to forward the replicas only to not yet visited nodes, but at least one replica is spread. In comparison to the traditional k-random search mechanism the proliferation is more efficient regarding produced messages.

4. Describing and Calculating the Multimedia Lifecycle

In this chapter a Video Notation (ViNo) is introduced that allows for the description of the whole video life cycle from request to delivery until presentation [1]. We concentrate on new usage patterns, where the videos are not accessed sequentially. As an example, doctors that record videos of their surgeries want to browse to specific scenes (see [67]). They might be interested in comparing similar scenes, play them in parallel and create sub-sequences or extract single images.

We therefore introduce the notion of *non-sequential media access*, where we consider that a video consists of a set of elementary video units. The size of such a unit is application dependent. If a system manages large videos, a unit might consist of a scene. If a system has to handle mainly short videos, it is not necessary to further split them up. The unit model, however, does not only cover videos. A unit might consist of arbitrary continuous media in combination with metadata, text, URLs, etc. Thus, we actually speak of multimedia units. The advantage of such a model is that users can compose their own multimedia presentation consisting of a number of arbitrary units. ViNo is the tool that helps the user by offering sequential and parallel operators.

The goal of ViNo is, however, not only to provide a tool for the formulation of presentations, the usage of ViNo for other purposes can be motivated by the following two reasons:

ViNo is *flexible*. By breaking with the traditional sequential pattern of continuous media, innovative possibilities in handling video systems appear. E.g., the usage of ViNo allows for a certain programmability of a multimedia delivery system.

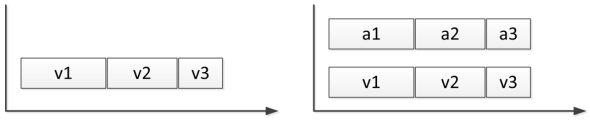


Figure 4.1.: Sequential and parallel time line of a SMIL presentation

ViNo allows for *simplification*. With the usage of ViNo whole systems can be described and explained with a short and simple notation. ViNo includes the specification of required and provided QoS.

4.1. Related Work

Several languages target either the description of transport (including QoS negotiation) or the description of video presentation. In the following an overview of both categories is given.

4.1.1. Presentation

The Synchronized Multimedia Integration Language (SMIL)[68] is one of the most popular XML-based languages to express multimedia presentations. Specific players are needed to convert the SMIL expressions to actual presentations.

A SMIL presentation consists of a *smil* element, a *head* and a *body* element. The head element contains meta information, layout information and author defined content control. In other words, everything not related to the temporal behavior of the presentation. The body element contains all information of the temporal behavior. SMIL allows for sequential timing, parallel timing and exclusive timing. The exclusive time container is similar to the parallel time container, it allows for defining several objects in parallel. But only one of the objects is selected for presentation. This pattern might be used for video sub-titles, where a number of sub-titles are defined, but only one is shown to the user. Within these time containers sub-elements can be placed with further attributes, such as duration and repetition. In Figure 4.1 we show example time lines for sequential and parallel presentations. In the first time line three videos are played sequentially. The second time line shows the parallel presentation of the videos and the corresponding audio files. The corresponding SMIL descriptions are shown Listings 4.1 and 4.2.

```
1  <smil xmlns="http://www.w3.org/ns/SMIL">
2    <head> </head>
3    <body>
4      <seq>
5          <video dur="10s" src="movie1.mpg" />
6          <video dur="10s" src="movie2.mpg" />
7          <video dur="5s" src="movie3.mpg" />
8      </seq>
9    </body>
10 </smil>
```

Listing 4.1: Sequential presentation of 3 videos

```
1  <smil xmlns="http://www.w3.org/ns/SMIL" >
2    <head> </head>
3    <body>
4       <par>
5          <video begin="0s" dur="10s" src="movie1.mpg" />
6          <audio begin="0s" dur="10s" src="sound1.au" />
7          <video begin="10s" dur="10s" src="movie2.mpg" />
8          <audio begin="10s" dur="10s" src="sound2.au" />
9          <video begin="20s" dur="5s" src="movie3.mpg" />
10         <audio begin="20s" dur="5s" src="sound3.au" />
11     </par>
12   </body>
13 </smil>
```

Listing 4.2: Parallel presentation of 3 videos and 3 audio files

4.1.2. Transport

QoS languages have been defined to help a user or application to specify requirements and to formally define actions for recovery if the given requirements are not met. In [69] Jin and Nahrstedt give an overview and classification of QoS languages, which are categorized into user-layer QoS, application-layer QoS and resource-layer QoS. User-layer QoS is an abstract specification of quality expectations by a user. Later, the application translates these QoS requirements to quantitative parameters. These parameters are resource dependent, and therefore have to be specified by the resource layer.

A user-layer QoS example is INDEX [70], an architecture that helps translating the user's preferences to more specific network-related QoS. A user can define the service quality and with that the price of the service. The task of the INDEX architecture is to optimize the cost-performance relation for the users. The user-layer parameters are translated in the lower layers to quantitative QoS parameters based on which the appropriate service provider is selected.

HQML [71] is an example for application-layer QoS, which is based on XML. Thus, it follows the mark-up based paradigm as defined by Jin and Nahrstedt in [69]. Developers can define their own QoS related tags and their own adaptation rules to switch between tags. Another example is QML [72], which follows the object-oriented paradigm, allows for QoS refinement by inheritance. At the lowest level one can define contracts with different dimensions at design time. The contracts can be grouped to contract types, which can be further combined to profiles. E.g., a server has to have high reliability and availability. To describe this, the profile is called server, and two contract types reliability and availability are defined. The contract types can be further refined to contracts with any specification needed.

RSL is a resource-layer QoS language [73], which allows for meta-level specification of resources. RSL is part of the Globus project that uses this language through all layers. At the application level a QoS requirement is formulated in RSL, then it is translated to a lower-level expression, which even defines which resource manager will handle the particular request.

Fine grained resource-layer QoS specification allows for specifying in detail, which resources are required. Examples are IntServ [74] and DiffServ[75]. IntServ uses the

Resource Reservation Protocol (RSVP)[76]. For each data stream resources are reserved along the transport path. However, the receiver is responsible for the initiation of the resource reservation. DiffSev simplifies the process of QoS aware distribution by concentrating on packets instead of streams. At the network edges the IP service header field is set, which can be used for routing decisions in the network core.

Another language, not discussed by Jin and Nahrstedt is QL designed by Blair et al., which is used in [77]. It is a raw language for time-based QoS specification, which includes the semantics of the basic QoS types delay, jitter, bandwidth, etc. Its simplicity allows for short descriptions of QoS requirements.

A QoS integration into a modern programming language is MMC# (see [78],[79]). MMC# is a QL based QoS extension of C#, which provides automatic QoS requirement checking. However, it requires the setup of the MMC# specific compiler.

The network calculus goes beyond QoS specification, by allowing the calculation of QoS in a network [80], [81]. The network calculus follows an input/output system of networks, sub-networks or nodes. The input is the data put into the network and the output the data to be consumed. The task of the network calculus is to indentify the worst case traffic described by the deterministic arrival curve concept. The arrival curve denotes the worst case traffic. Le Boudec further defines in [80] service curves, which denote what services a system offers. Both curves are used to determine upper bounds of QoS metrics, such as delay, output bounds, etc.

4.2. The Video Notation (ViNo)

In the following the Video Notation (ViNo) is defined and described[1]. Note that the full EBNF specification can be found in Appendix A.

Definition 1 *A composition is an expression defined inductively by these rules:*

1. *A single video unit is a composition.*
2. *Let $c_1, c_2, ..., c_n$ with $n \geq 2$ be compositions, which have already been defined. Then, the following expressions are compositions, too:*
   a) $[c_1 \,||\, c_2 \,||\, \ldots \,||\, c_n]$ is called a parallel composition.

[1] This section is adapted from [1]

b) $(c_1 \leftarrow_{Q_1} c_2 \leftarrow_{Q_2} \cdots \leftarrow_{Q_{n-1}} c_n)$ *is called a sequential composition. A symbol* Q_i*, where* $i = 1, \ldots, n-1$*, represents an optional QoS parameter and may be omitted.*

u_i $(i \geq 1)$ always denotes a single video unit. The brackets or parentheses of a parallel or a sequential composition c, respectively, may be omitted if c does not appear as proper subexpression of a composition. So both $[u_1 \,||\, u_2]$ and $u_1 \,||\, u_2$ are valid compositions on their own, but $u_1 \,||\, u_2 \leftarrow u_3$ is not.

In the following the semantics of ViNo are described. Here, the context is defined to be video transport. However, it is also possible to define semantics of ViNo for other purposes (e.g. video playback or video queries).

Definition 2 *Semantics.*

1. *If* $c = c_1 \,||\, c_2$ *for some compositions* c_1 *and* c_2*, then the transport of* c *starts as soon as* c_1 *or* c_2 *starts, whatever is earlier; and it is finished when the transport of both* c_1 *and* c_2 *is completed.*

2. *If* $c = c_1 \leftarrow_Q c_2$ *then the transmission of* c_2 *must not start before the completion of* c_1*; the QoS predicate* Q *applies to the time period between completion of* c_1 *and completion of* c_2*.*

3. *The semantics of* $c = c_1 \leftarrow_{Q_1} c_2 \leftarrow_{Q_2} c_3$ *is defined as that of* $(c_1 \leftarrow_{Q_1} c_2) \leftarrow_{Q_2} c_3$*.*

We consider two compositions c_1 and c_2 as *equivalent* if they lead to the same semantics according to Definition 2. We then write $c_1 = c_2$. It is easy to check that the following equations hold:

$$[c_1 \,||\, c_2] \,||\, c_3 = c_1 \,||\, [c_2 \,||\, c_3] \tag{4.1}$$

$$[c_1 \,||\, c_2] = [c_2 \,||\, c_1] \tag{4.2}$$

$$(c_1 \leftarrow c_2) \leftarrow c_3 = c_1 \leftarrow (c_2 \leftarrow c_3) \tag{4.3}$$

$$(c_1 \leftarrow_{Q_1} c_2) \leftarrow_{Q_2} c_3 = c_1 \leftarrow_{Q_1 \circ Q_2} (c_2 \leftarrow c_3) \tag{4.4}$$

where c_1, c_2, c_3 are arbitrary compositions and $Q_1 \circ Q_2$ means a suitable combination of both QoS predicates Q_1 and Q_2, e.g. the sum if Q_1 and Q_2 refers to maximal delay.

Note that according to Equation 4.1 a parallel composition c is an associative binary operation, so the semantics of c are well defined. The same applies to sequential compositions without QoS predicate.

Definition 3 *The null unit u_0 is a video unit of length 0 (empty).*

The null unit u_0 serves as a *dummy* composition (in a similar way as dummy targets are used to express side-effects in functional languages). The following properties apply:

- $u_0 \,||\, c_1 = c_1$
- $u_0 \leftarrow c_1 = c_1$
- $c_1 \leftarrow u_0 = c_1$

4.2.1. Simple examples

In the following two different examples are discussed for showing the application of ViNo. In both cases the units are located at the origin server and have to be transported through the network. The basis is a video architecture consisting of an origin server and five proxies. We do not consider QoS yet.

ViNo can be used to express a user request. The request can represent the required presentation of the units. In our example the restriction is set to a sequential order of the three units and can be expressed as follows.

$$r = u_1 \leftarrow u_2 \leftarrow u_3$$

The request is part of the delivery, which is divided into a number of stages. After the request, stages describe the hops traveled by the units. The final stage is the presentation.

For sake of simplicity we assume for the current examples that the presentation starts after full download of all units.

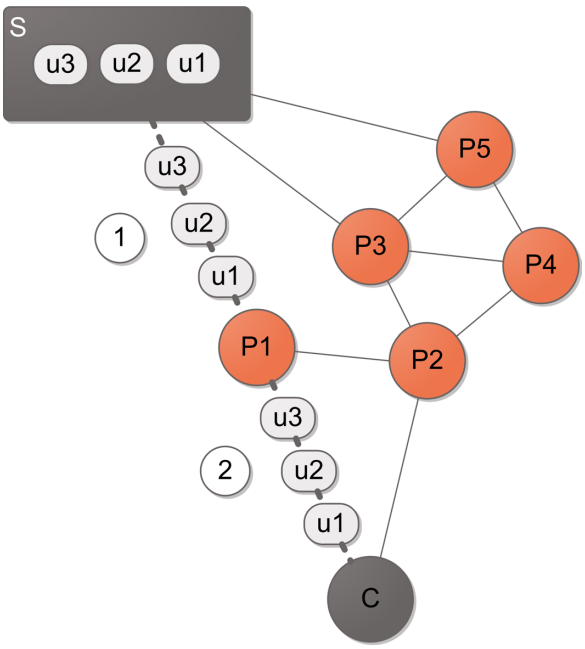


Figure 4.2.: Sample video delivery system with one origin server S, five proxies $P1 - P5$, and one client C, where the client wants to get u_1, u_2, u_3 sequentially

Sequential Download from Origin Server

In this example the proxies act as caches. The units are located at the origin server and may be replicated at the proxy servers. Figure 4.2 depicts the situation. On the shortest path the transport would take two hops, from S to P1 and from P1 to the client C.

The timely characteristics can be described as shown in Table 4.1. The columns represent time slots and the lines are stages. For sake of simplicity it is assumed that one unit takes one time slot for the transport over one hop. In the first stage everything is downloaded to P1, which takes three time slots. After finishing the download P1 forwards the units in stage s_2 to the client, which starts to present the units after all units are downloaded. The completion of the transport including presentation takes nine time slots.

In the corresponding ViNo notation it can be seen that it is a short hand for the tabular representation of the time line. For better readability optional round brackets are used to differ between the stages (see Definition 2).

$$c = (u_1 \leftarrow u_2 \leftarrow u_3) \leftarrow (u_1 \leftarrow u_2 \leftarrow u_3) \leftarrow (u_1 \leftarrow u_2 \leftarrow u_3)$$

t	1	2	3	4	5	6	7	8	9
s_1	u_1	u_2	u_3						
s_2				u_1	u_2	u_3			
p							u_1	u_2	u_3

Table 4.1.: Temporal evolution of the video delivery including presentation

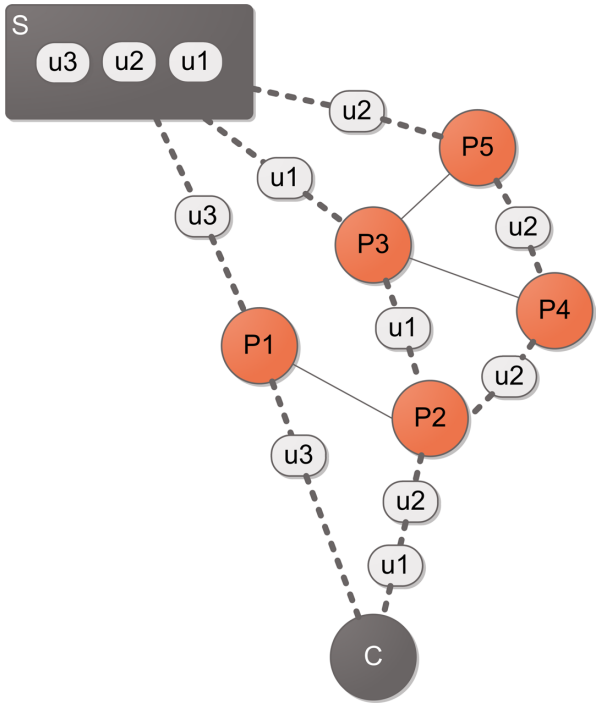


Figure 4.3.: Sample video delivery system with one origin server S, five proxies $P1 - P5$, and one client C, where the download is done over a number of proxies

We can even simplify the ViNo expression before. Since the single stages are equal (according to Definition 2), the resulting ViNo expression is denoted as follows:

$$c = s_1 \leftarrow s_2 \leftarrow p$$
$$where\ s_1 = s_2 = p = u_1 \leftarrow u_2 \leftarrow u_3$$

Mixed Download from Origin Server

In this example shown in Figure 4.3 the server spreads the units over the network. The three units travel in parallel from the server towards the client. Unit u_2 has the longest path over proxies P_5, P_4 and P_2. The number of stages increases remarkably, because the number of proxies involved increases. Stage s_1 therefore describes the transport of unit u_3 from the server to proxy P_1. In parallel to that units u_1 and u_2 are

t	1	2	3	4	5	6	7
s_1	u_3						
s_2	u_2						
s_3	u_1						
s_4		u_3					
s_5		u_2					
s_6		u_1					
s_7			u_2				
s_8			u_1	u_2			
p					u_1	u_2	u_3

Table 4.2.: Temporal evolution of the video delivery including presentation

transported to P_3 and P_5. One can say for each link traveled one stage exists plus the presentation. In this example we need 8 stages plus 1 presentation stage. The tabular representation is shown in Table 4.2. One can see immediately that the download takes less time as for Example 1, because of the parallel delivery. Additionally, more nodes can cache the content than before. Stage s_8 subsumes the fact that during the forwarding of u_1 to the client unit u_2 is not available yet, but immediately after download it is forwarded to the client. Note that this works because it is assumed that the transport of u_1 takes the same time as the transport of u_2. Otherwise, stage s_8 has to be split up into two stages.

The corresponding ViNo expression is shown in the following expression 4.5. Stepwise we simplify the ViNo expressions, from units to stages, to compositions. If a system designer would have to decide about a unit forwarding strategy a short ViNo description helps for comparison.

$$\begin{aligned} c = [u_3||u_2||u_1] \leftarrow [u_3||u_2||u_1] \leftarrow [u_2||(u_1 \leftarrow u_2)] \leftarrow (u_1 \leftarrow u_2 \leftarrow u_3) \\ = [s_1||s_2||s_3] \leftarrow [s_4||s_5||s_6] \leftarrow [s_7||s_8] \leftarrow p \\ = c_1 \leftarrow c_2 \leftarrow c_3 \leftarrow p \end{aligned} \quad (4.5)$$

4.2.2. Introducing QoS

To specify a request a client has only to provide information about the required video units and whether these units have to arrive in order (e.g., at the player). For example, a user may express "I want to download units x and y, the order does not matter" as $u_x \,||\, u_y$. Note that this does not mean that the units have to be delivered

in parallel. A user who wants to watch the units using a video player would be more specific: "I want to watch unit x and then unit y, and unit x has to arrive within the next 30 seconds". This request can be expressed as $u_0 \leftarrow_{D=30sec} u_x \leftarrow u_y$, where u_0 is the null unit needed only to express the required delay for unit u_x.

In the following the usage of QoS parameters is demonstrated for expressing video unit transport, i.e., the provided QoS. If ViNo is used in a different context, the semantics of QoS annotations may differ and need to be clarified prior to any calculations based on ViNo expressions. In this section examples are given for bandwidth, delay and jitter as QoS parameters. These QoS parameters are further used as input for a delay calculation of a composition.

The notation and semantics of the QoS parameters are derived from the QoS language QL [77]. QL is based on events like the reception and the sending of messages. It uses a function τ for mapping events to points in time. ViNo concentrates on the event of *receiving a composition c* at a given network node or video display. This event occurs as soon as all video units referenced by c have been received completely. The corresponding point in time can be denoted as $\tau(c)$.

Definition 4 *We define a recursive function delay to calculate a delay bound for a QoS-annotated ViNo transport description c:*

1. *The null unit causes no delay:* $delay(u_0) = 0$.

2. *If* $c = c_1 \leftarrow_Q u$ *for some composition* c_1 *and a video unit* u*, then*

$$delay(c) = delay(c_1) + delay(u, Q)$$

   where $delay(u, Q)$ *is defined to be the delay* $\tau(u) - \tau(c_1)$ *assuming a given QoS parameter* Q *(trivial case of recursion).*

3. *If* $c = c_1 \leftarrow c_2$ *for compositions* c_1 *and* c_2*, then the delay bound is computed recursively as:*

$$delay(c) = delay(c_1) + delay(c_2)$$

   For delay calculations, we assume that the transmission of c_2 *occurs as soon as possible after the transmission of* c_1*, which is expressed by omitting the QoS parameter.*

4. *If* $c = c_1 \,||\, c_2$ *for compositions* c_1 *and* c_2*, then the delay bound is computed recursively as:*

$$delay(c) = max(delay(c_1), delay(c_2))$$

Note that the *delay* function is defined only on a subset of all possible ViNo expressions.

The following two expression types will occur frequently in the subsequent examples, so a separate notation for the corresponding delay bounds is defined:

- If $c = u_0 \leftarrow_{Q_1} u_1 \leftarrow_{Q_2} \cdots \leftarrow_{Q_n} u_n$ for video units u_i (u_0 is the null unit), then

$$delay(c) = \sum_{i=0}^{n} delay(u_i, Q_i) = delay(u_1, \ldots, u_n, Q_1, \ldots, Q_n, seq) \tag{4.6}$$

  where the last term introduces a new notation for indicating sequential calculations.

- If $c = (u_0 \leftarrow_{Q_1} u_1) \,||\, \ldots \,||\, (u_n \leftarrow_{Q_n} u_n)$ for video units u_i (u_0 is the null unit), then

$$delay(c) = \max_{1 \le i \le n} (delay(u_i, Q_i)) = delay(u_1, \ldots, u_n, Q_1, \ldots, Q_n, par) \tag{4.7}$$

  where the last term introduces a new notation for indicating parallel calculations.

Whether the *delay* function represents a lower or upper delay bound depends on the definition of the $delay(u, Q)$ values. In the following it is demonstrated how to define these values if the QoS parameters are given in terms of bandwidth, delay, or jitter, respectively.

Bandwidth

By $Q = BW$ it is expressed that a given bandwidth BW is available for transmission. The delay of transmitting a video unit u is then defined as:

$$delay(u, BW) = \frac{size(u)}{BW}$$

The *delay* function therefore computes a *lower bound* of the end-to-end delay corresponding to a given ViNo expression. We assume that video units transmitted in parallel according to some parallel ViNo composition use separate links, so that the

available bandwidth is not reduced by parallel transmissions.

The lower delay bound of sequential and parallel compositions calculate as shown in equation 4.6 and equation 4.7.

$$delay(u_1, \ldots, u_n, BW_1, \ldots, BW_n, seq) = \sum_{i=1}^{n} \frac{size(u_i)}{BW_i} \tag{4.8}$$

$$delay(u_1, \ldots, u_n, BW_1, \ldots, BW_n, par) = \max_{1 \leq i \leq n} \left(\frac{size(u_i)}{BW_i} \right) \tag{4.9}$$

Delay

By $Q = D$ it is expressed that transmission yields a given delay D. The delay of transmitting a video unit u is defined as:

$$delay(u, D) = D$$

If all delays occurring in a video delivery system are expressed as provided QoS parameters of a corresponding ViNo composition and if the composition is an appropriate model of the system, the calculated end-to-end delay value should be accurate. However, for practical purposes, the ViNo composition is constructed to provide an *upper delay bound* only.

Jitter

According to the QoS language QL, jitter can be defined by specifying *lower and upper delay bounds* (D_{min}, D_{max}). To calculate the jitter of a given video transport system described by an appropriate ViNo expression, we therefore just need to apply the *delay* function to both bounds separately. We obtain two functions $delay_{min}$ and $delay_{max}$ with appropriate definitions of delay bounds for transmitting a video unit u:

$$delay_{min}(u, D_{min}) = D_{min}$$
$$delay_{max}(u, D_{max}) = D_{max}$$

The jitter of a ViNo composition c is then computed as $(delay_{min}(c), delay_{max}(c))$.

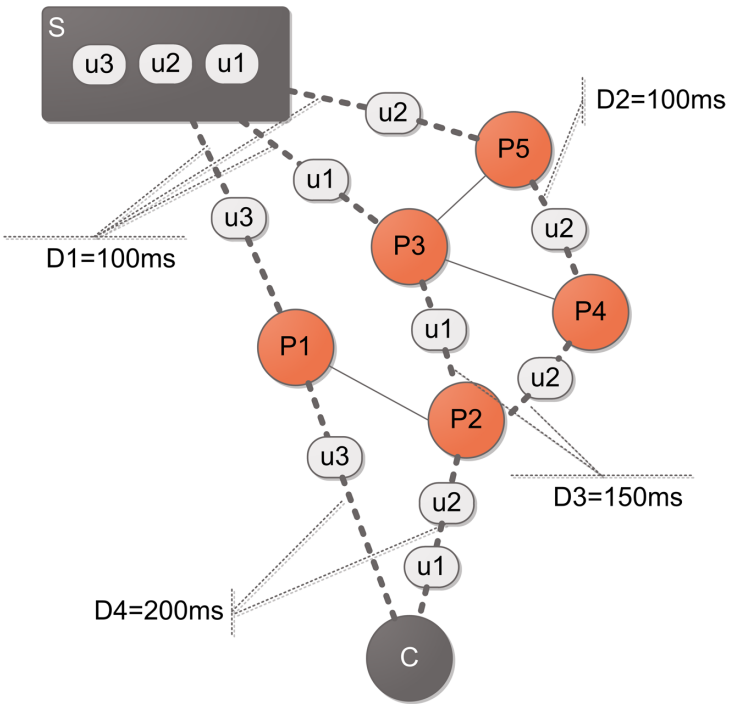


Figure 4.4.: Sample network with delay (D) as QoS parameter

4.2.3. Simple Examples with QoS

In the following it is demonstrated how the delay calculations can be applied. For this purpose we extend example 2 from Section 4.2.1. The adapted QoS parameters are shown in Figure 4.4. The ViNo expression of this example has to be extended to include the given QoS parameters such as shown in Expression 4.10. We want to calculate the pure transport, therefore we omit the presentation stage. The first null unit u_0 is needed to express that the delay applies for each unit of the parallel group. The second parallel group needs the longer version of the notation, since for each unit a different delay applies. Note that the parallel transmission is terminated if all units of its group are delivered. Additionally, since delay D_3 and D_4 differs one has to ensure that unit u_2 can only be transported to the client, if it is already at P_2. Therefore, the composition has to be changed, such that u_2 is delivered sequentially after the parallel transport of u_2 to P_2 and u_1 to C.

$$\begin{aligned} c = u_0 \leftarrow_{D_1} [u_1 \,||\, u_2 \,||\, u_3] \leftarrow [(u_0 \leftarrow_{D_3} u_1) \,||\, (u_0 \leftarrow_{D_2} u_2) \,||\, (u_0 \leftarrow_{D_4} u_3)] \\ \leftarrow [(u_0 \leftarrow_{D_3} u_2) \,||\, (u_0 \leftarrow_{D_4} u_1)] \leftarrow_{D_4} u_2 \end{aligned} \tag{4.10}$$

In shorter form, each parallel group can be seen as composition, thus we get the form

$c = c_1 \leftarrow c_2 \leftarrow c_3 \leftarrow u_2$. This notation further simplifies the delay calculation of composition c. If we apply Equation 4.7 and case 3 of Definition 4, the calculation can be described as follows:

$$
\begin{aligned}
delay(c_1) &= delay(u_1, u_2, u_3, D_1, par) = D_1 \\
delay(c_2) &= delay(u_1, u_2, u_3, D_3, D_2, D_4, par) = max(D_3, D_2, D_4) = D_4 \\
delay(c_3) &= delay(u_1, u_2, D_4, D_3, par) = max(D_3, D_4) = D_4 \\
delay(c) &= delay(c_1) + delay(c_2) + delay(c_3) + delay(u_2, D_4) = D_1 + D_4 + D_4 + D_4 \\
&= 100 + 200 + 200 + 200 = 700\text{ms}
\end{aligned}
$$

For each composition we calculate the delay to finally sum up the single results to the overall delay of 700 ms.

4.2.4. Introducing Wildcards

In dynamic systems, such as in wireless multi-hop networks and unstructured peer-to-peer systems, it is hard to manage a global index. ViNo allows the formulation of search queries, where it is not necessary to know the ID of a unit. The symbol ? can be used to denote a wildcard for such a not fully specified request. A request could look like ?||?, where a user wants to see two units (any units) in parallel. If the units should have specific characteristics the client can specify the request with the help of the QoS parameter preceded by the sequential operator (see Definition 1 and Definition 2). For example, a user wants to have two units in parallel, the first unit should be described by sun and summer and the second unit should be described by winter and should have a playback time of 5 s. This can be expressed by: $(u_0 \leftarrow_{tag=sun,tag=summer} ?)||(u_0 \leftarrow_{tag=winter,playbacktime=5s} ?)$. The syntax rules are specified in Definition 5.

Definition 5 *We define the usage of wildcards for formulating requests:*

- *The wildcard unit ? is a placeholder for a real unit.*
- *Characteristics of the wildcard unit can be specified as a list of key = value pairs.*
- *Characteristics are linked to a ? unit by using a preceding ← operator.*

Depending on the context in which ViNo is used, the characteristics of a wildcard may vary. The application has to define, which metadata are available and how the system manages the metadata. We leave the definition of the matching policy of wildcards and content, to allow for flexible usage of ViNo.

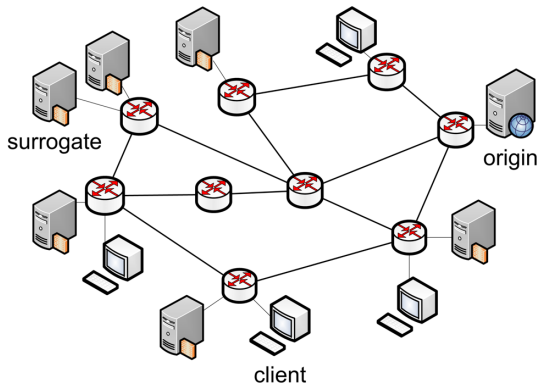


Figure 4.5.: Sample architecture using CDNsim [1]

4.3. Applicability of ViNo for Describing Content Delivery Networks

In this section[2] the potential of ViNo is shown as a tool for a quantitative analysis of existing delivery systems, such as *Content Delivery Networks* (CDN).

CDNs consist of origin servers supported by strategically placed surrogate servers to which the content is replicated and/or cached (see [82] [83]). In most of the commercially available CDNs, the content is passively pulled by surrogate servers. Usually, commercial CDNs are not available for research purposes. Even academic CDNs, which are available on PlanetLab, are treated as black boxes. For this reason, Stamos et al. [84] developed a simulation environment, called *CDNsim*, for large scale CDN simulations.

A GUI for configuring simulations is also part of CDNSim. CDNSim is an Omnet++ simulation and uses the INET Framework Library extension [85]. It covers all typical CDN functionality, such as DNS request redirection and LRU replacement. CDNSim supports different cooperation policies such as closest surrogate or random surrogate cooperation, but also simple non-cooperative behavior can be configured.

We use CDNSim to compare its results with the calculations made with ViNo. We perform two experiments. In the first experiment we simulate a CDN and measure the client end-to-end delay. At the same time we perform ViNo calculations and evaluate the differences between simulation and calculation. The second experiment

[2]This section is adapted from [1]

Parameter	Value
Router Topology	Waxman for 1,000 Routers
Link speed	200 Mbit/sec
Number of clients	100
Number of surrogate servers	100
Number of origin servers	1
Number of outgoing connections	1,000
Traffic	1,000,000 requests popularity's zipf = 1.0 expo mean interarrival time = 1
Resources	50,000 files max. size=100MB Zipf skew = 1.0 correlation between file size and popularity=0
Cache size	10 % - 109MB
Strategy on miss	download from origin

Table 4.3.: CDNsim configuration parameters [1]

shows a similar scenario, but we concentrate on file size differences, i.e., we want to show that different file sizes do not have an impact on the ViNo calculation.

We configured the CDN simulation as shown in Table 4.3. CDNSim includes sample traces and router topologies, which we also took for our simulation. For sake of simplicity we decided to use the non-cooperative policy for our experiments, i.e, if a requested object is not available at the client's surrogate server the request is forwarded to the origin server. However, all delay calculations can also be applied to the cooperative policies.

In this scenario a unit is defined as a part of a web page, with a fixed size of 1,500 bytes. This means that a web page of 4,500 bytes is divided into 3 units and is described as $u_1 \leftarrow u_2 \leftarrow u_3$. The link speed is 200 Mbits/sec, which results in a bandwidth BW of 16,666 units/sec.

We performed two types of ViNo calculations. The first type called *ViNo generic* assumes that a hit requires one hop for downloading a unit and a miss requires two hops, when the content has to be brought from the origin server. The second type called *ViNo routers* knows the network structure and has information about the number of routers between client and surrogate and between surrogate and origin

server. If we assume that the routes do not change, the calculations can be done precisely. The calculations represent the time a transport takes at minimum, i.e., it is the lower-bound transport delay.

Experiment 1

In this setting we take for each client one random web site, where the web sites are of similar size (in average 3,9 KB). Furthermore, each web site is not available at the connected surrogate server, i.e., is a miss. Thus, we can see the differences between ViNo routers and ViNo generic. We further measure the delay in the simulation and compare it with the ViNo calculations.

To show an example calculation for *ViNo routers* we choose a random client with the ID $c1009$. The client connects to the surrogate server with ID $s1199$ in 4 hops, and the surrogate is connected to the origin within 5 hops. Thus, a miss means a transport over 9 hops from the origin server. This client downloads the object with ID 13, which has a size of 5 units. In ViNo one stage consisting of these 5 units can be described as $c_i = u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3 \leftarrow_{BW} u_4 \leftarrow_{BW} u_5$. The composition c for ViNo routers is a sequential composition of stages 1-9 (one stage per hop).

$$\mathrm{h} = 9, \mathrm{BW} = 16,666 u/sec$$

$$delay(c) = \sum_{i=1}^{h} delay(c_i) = h * delay(u_1, ..., u_5, BW, ..., BW, seq) = 2.7\ ms$$

The *ViNo generic* calculation has no detailed knowledge of the routers and the same situation would be calculated as:

$$\mathrm{h} = 2, \mathrm{BW} = 16,666 u/sec$$

$$delay(c) = \sum_{i=1}^{h} delay(c_i) = h * delay(u_1, ..., u_5, BW, ..., BW, seq) = 0.6\ ms$$

The difference between ViNo generic and ViNo routers is the detail of information, which is often not available.

In Figure 4.6 we subsume the simulated and calculated delay for one unit by client. One can see that both ViNo routers and ViNo generic estimate the delay pattern of the measured values well. However, we see clearly that the additional information of

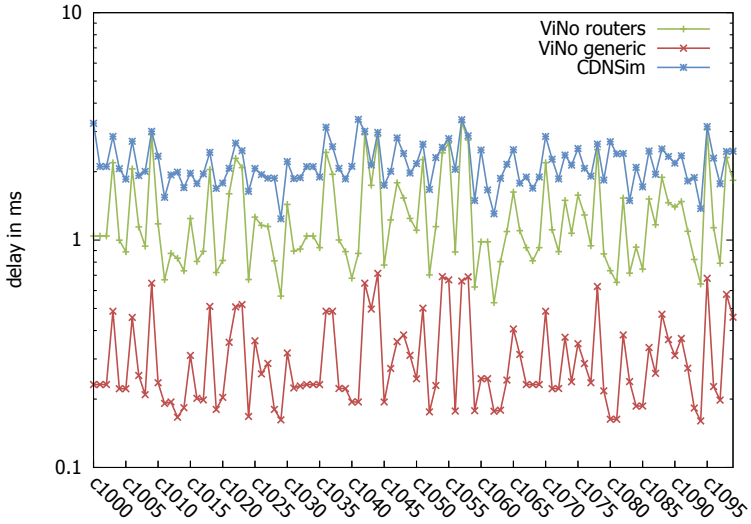


Figure 4.6.: Delay comparison between ViNo and CDNSim of one miss web site chosen per client[1]

ViNo routers improves the result. Since the size of the units is very similar, the delay is stable. In a real environment the hit rate and miss rate have to be estimated, to draw conclusions on the performance of the overall system.

Experiment 2

In this experiment we consider all file sizes of one client. The file sizes are Zipf-distributed, which means that 80 % of the overall objects' size is represented by 20 % of the objects. This fact has a huge impact on the surrogates' cache size. In CDNSim one cache was able to store 109 MBs, which lead to hit rates of around 80 %. The reason is that most surrogates handle small files and the cache misses only occurred in the beginning of the simulation until all objects were loaded from the origin server (i.e., the actual cache size was around 100 %).

We picked client c1020 randomly, and investigate the delay differences of the first 200 requested units. The ViNo router calculated delay is shown in Figure 4.7 (log scale). One can see that the calculations do not always represent the lower bound of the simulated duration. One extreme case is shown for the object with ID 637 (the peak in Figure 4.7), which has a size of 24 MB (16,666 units). At this point of the

simulation the object was not present at the surrogate, thus it had to be downloaded from the origin with a distance of 8 hops. The measured value was 3,000 ms. The calculations with ViNo routers are provided below:

$$delay(c) = \sum_{i=1}^{8} delay(c_i) = 8 * delay(u_1, ..., u_{16666}, BW, ..., BW, seq) = 8,000ms$$

This effect appears for files that exceed the size of 10 KB, which are routed in a different way than smaller files (as in experiment 1). Larger files are split up and are routed in parallel over several paths. Therefore, the transport is a mixture of parallel and sequential compositions and not purely sequential as assumed before. Since the routing algorithm is part of the INET Framework and the paths are not predictable with reasonable effort, we cannot provide a more detailed calculation. However, the router based calculations might represent the worst case delay well, if the routing path is always the same.

The generic calculations are always representing the lower bound of the duration as shown in Figure 4.8, since the surrogate downloads the complete web site before forwarding it to the client. In comparison to ViNo routers the ViNo generic result is 2,000 ms for the object with ID 637.

Thus, the generic case represents the larger file downloads better and the router based calculations represent smaller file downloads better. Which type of calculation is finally taken depends on the knowledge of the architecture and on the purpose of the analysis.

Using ViNo to Investigate Performance Improvements

The efficiency of CDNs and caches is usually compared by measuring the hit rate. ViNo can also be used to analyze the impact of new strategies on the hit rate. One question could be how much delay improvement by the pipelining of units could be reached.

For this investigation we took ten random surrogate servers out of the simulation and calculated the delay reduction as shown in Table 4.4. It can be seen that the number of units (i.e., the size of the original objects) to serve vary a lot. E.g., surrogate s1191 only serves 19 objects. Surrogates s1158 and s1164 serve almost the same amount of units, but the number of objects differs by a factor of 10.

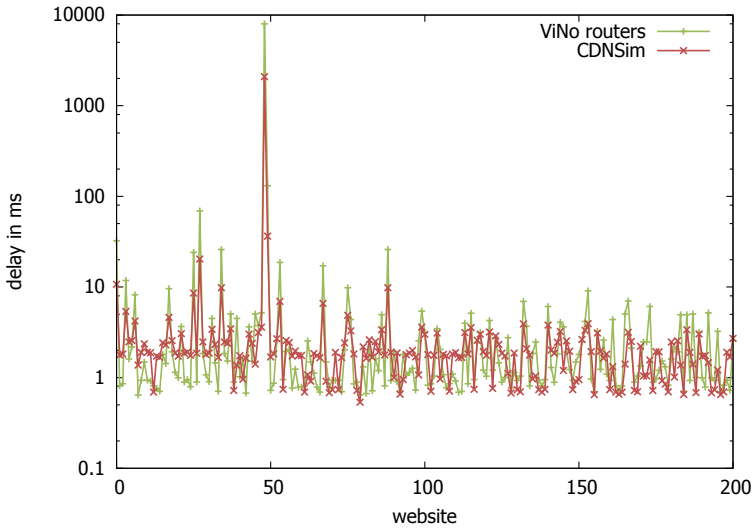


Figure 4.7.: Comparing ViNo routers and CDNSim - delay of the first 200 web sites of client c1020 [1]

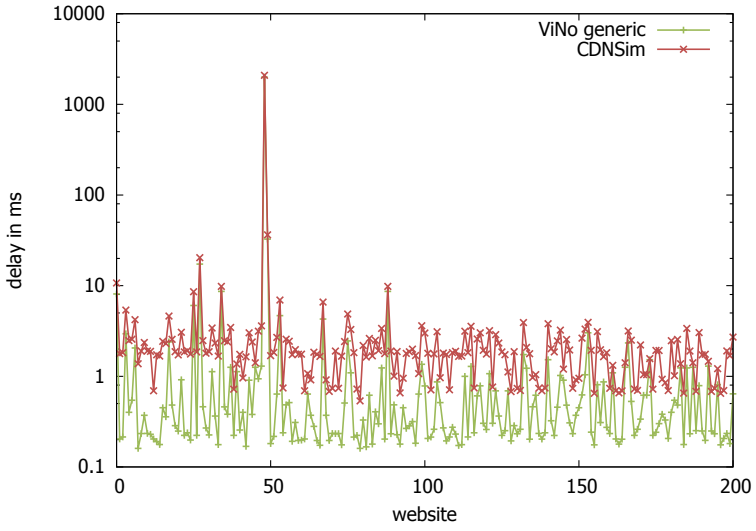


Figure 4.8.: Comparing ViNo generic and CDNSim - delay of the first 200 web sites of client c1020 [1]

SID	no. clients	no. objects	no. units	delay sequential	delay pipeline
s1110	2	2158	26290	3.155	1.577
s1132	1	475	4033	0.484	0.242
s1140	3	46	210	0.025	0.013
s1158	1	214	18352	2.203	1.101
s1164	3	2632	18597	2.232	1.116
s1188	4	3007	49100	5.892	2.946
s1191	2	19	69	0.008	0.004
s1194	2	6658	77183	9.262	4.631
s1196	4	3355	56281	6.754	3.377
s1198	6	3291	39795	4.775	2.388

Table 4.4.: Surrogates service during simulation, the average sequential delay and the delay improvement if pipelining is used [1]

A strategy for the CDN provider could be to apply pipelined transport on a miss, i.e., a surrogate forwards a unit immediately after download from the origin. For a web site that consists of three units this is described as:

$$c = (u_0 \leftarrow_{BW} u_1) \leftarrow [(u_0 \leftarrow_{BW} u_2)||(u_0 \leftarrow_{BW} u_1)] \\ \leftarrow [(u_0 \leftarrow_{BW} u_3)||(u_0 \leftarrow_{BW} u_2)] \leftarrow (u_0 \leftarrow_{BW} u_3)$$

In comparison to $c_1 \leftarrow c_2 = (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3) \leftarrow (u_0 \leftarrow_{BW} u_1 \leftarrow_{BW} u_2 \leftarrow_{BW} u_3)$ for the pure sequential transport. The delay for the pipelined composition is calculated as the sum of all sub-compositions (i.e., $c = c_1 \leftarrow c_2 \leftarrow c_3 \leftarrow c_4$).

$$\begin{aligned} & delay(c) \\ & = \sum_{i=1}^{4} delay(c_i) \\ & = delay(u_1, BW) \\ & + \max(delay(u_2, BW), delay(u_1, BW)) \\ & + \max(delay(u_3, BW), delay(u_2, BW)) \\ & + delay(u_3, BW) \\ & = \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} = 0.24ms \end{aligned}$$

If the units would be transported sequentially as in $c_1 \leftarrow c_2$ the delay would be

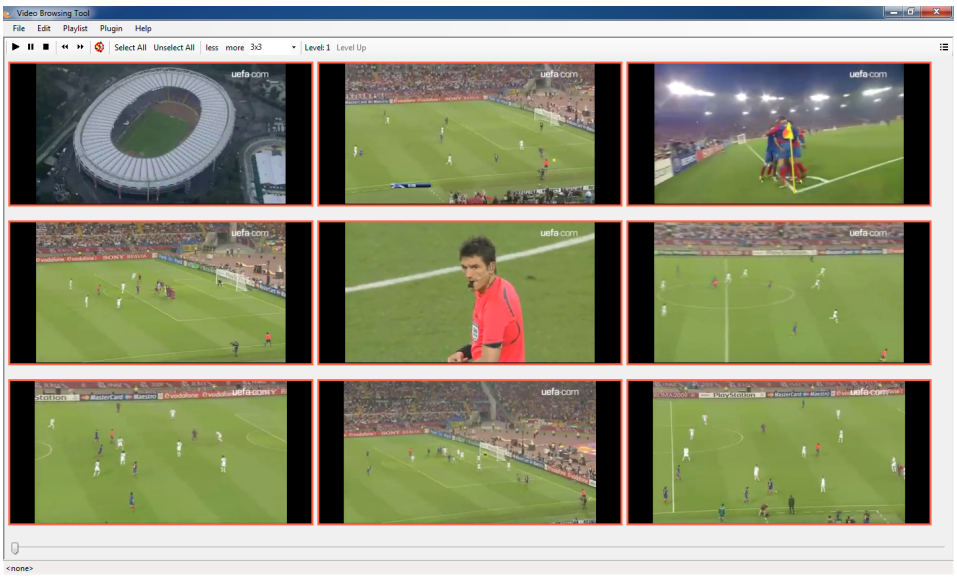


Figure 4.9.: Video Browser with 3x3 aligned units [4]

calculated as the sum of the sub delays, i.e:

$$
\begin{aligned}
& delay(c) \\
& = delay(u_1, u_2, u_3, BW, ..., BW, seq) \\
& + delay(u_1, u_2, u_3, BW, ..., BW, seq) \\
& = \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} + \frac{1}{BW} \\
& = 0.36ms
\end{aligned}
$$

The transport would need 3+3=6 time slots. The pipelined transport reduces the number of time slots to 4.

If the surrogates analyzed before used pipelined transport on a miss the delay would reduce in comparison to a sequential transport as shown in Table 4.4. The larger the number of objects to serve, the better pipelining supports the surrogates. Popularity-based handling of content could further increase the performance of the single surrogates.

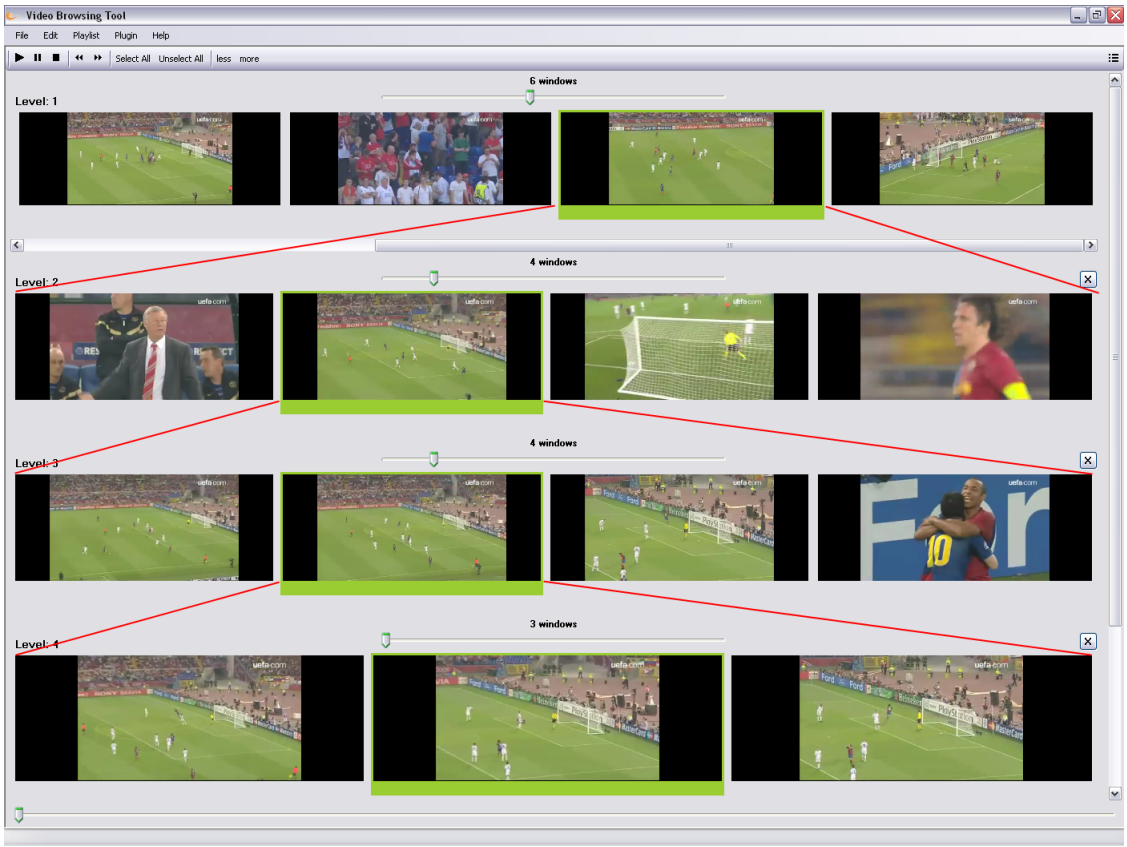


Figure 4.10.: Video Browser with tree-like presentation of units [4]

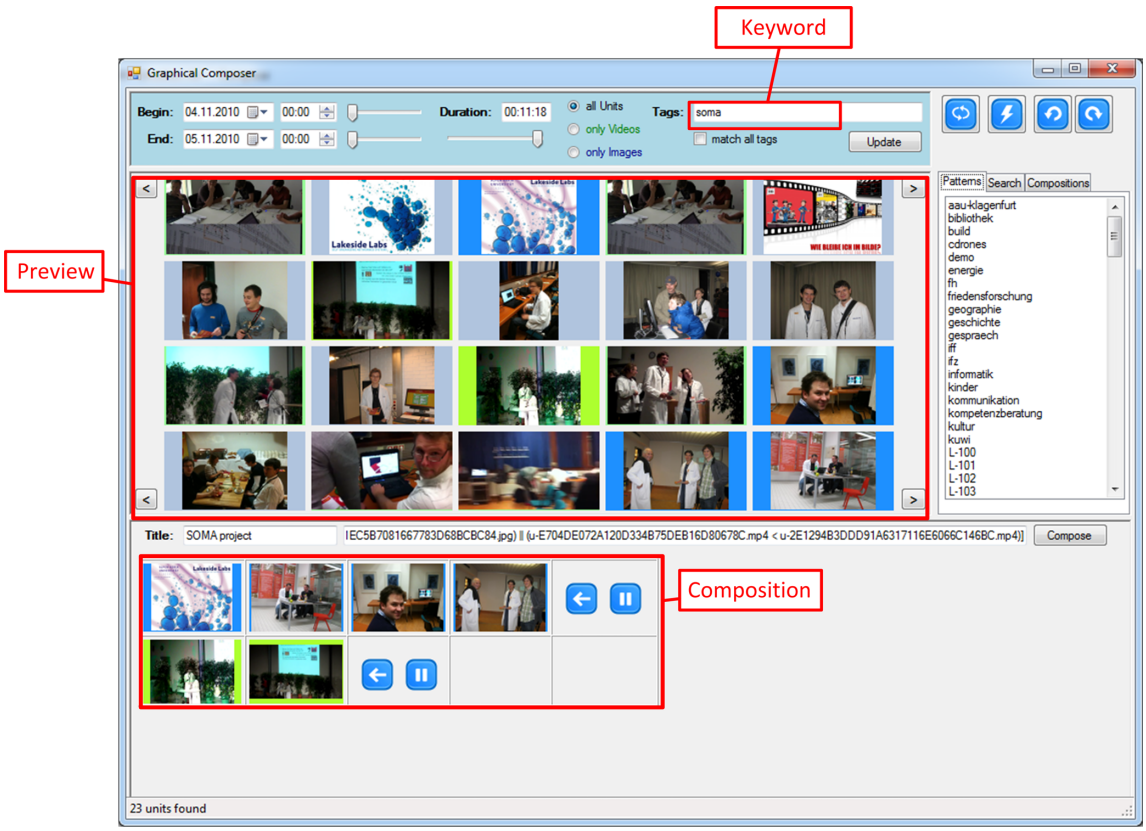


Figure 4.11.: Video Browser request interface using ViNo [5]

4.4. Applicability of ViNo for Requests and Video Presentation

Within SOMA, ViNo is used as the interface between user and delivery layer. A client can specify its queries by using ViNo, which can be at the same time be used as presentation description. In [4] a ViNo driven video browser is introduced. This video browser divides videos into video units and can present these units in parallel. In Figure 4.9 this player is shown, which offers a tabular presentation of nine units. The tabular presentation consists of three rows with three videos each. This presentation helps the user to navigate through the video, because each unit represents one ninth of the time span of the whole video. The shown example can be expressed by using ViNo. If we assume that u_i represents the corresponding unit IDs the expression $[u1||u2||u3]|[u4||u5||u6]|[u7||u8||u9]$ defines the current presentation. Each parallel group [] represents one row, the whole expression tells the player that the playback for all units starts immediately. During this presentation the user can interact with the video browser. If a user clicks on one of the video units, the time

span of the video is split again and a row of parallel units is presented, the user might navigate further and gets the next parallel rows (see Figure 4.10). The new presentation can be seen as a tree that can be navigated until the smallest time span is reached that still contains units. A new level of the tree terminates the presentation of the higher level presentation. Thus, each row has its own corresponding ViNo expression, which is $[u_w||u_x||u_y||u_z]$. The parallel and tree-based presentation modes are more complicated than the typical sequential presentation of videos and ViNo supports the player in solving this task.

Further extensions target smart summaries of social events [5]. In this work Flickr and YouTube are the source platform for videos and photos. Search queries are mapped to HTTP requests and sent to the platforms. Then, the content is downloaded and categorized before the presentation. If this event summarization is integrated into the SOMA architecture, the requests can be formulated by using ViNo. A scenario could be that a user searches for keywords (e.g., soma) and gets locally stored preview images presented. In ViNo such a request is declared as $u_0 \leftarrow_{tag=soma}$ [[?||?||?||?||?]||[?||?||?||?||?]||[?||?||?||?||?]||[?||?||?||?||?]], which means that a user wants to see four rows of five units each. The application translates the wild cards to preview units with the tag *soma*. The search result is shown in the upper part (preview) of the video browser interface as shown in Figure 4.11. The user can manually compose a presentation of the resulting preview units as shown in the lower part of the same figure. The user can compose sequential presentations as columns in one row, each parallel presentation starts with a new row. If assuming that the user chose the previews that match to real units u_1 to u_6 the corresponding ViNo expression for the example shown in Figure 4.11 is $(u_1 \leftarrow u_2 \leftarrow u_3 \leftarrow u_4)||(u_5 \leftarrow u_6)$. The delivery layer collects the required units with the given parameters.

4.5. Summary and Discussion

In this chapter we introduced the Video Notation (ViNo), which can be seen as a language framework for specifying the timely behavior of all phases of the life cycle of a video. It can be used as an interface between user and delivery, as a decision support for system designers and as short description language for the delivery of content. Two extensions allow for defining required QoS, calculate provided end-to-end QoS, and support keyword search.

Two case studies were made to show the applicability of ViNo. In the first scenario we compared ViNo calculations of different granularity with a simulation of a CDN. The results show that ViNo cannot fully substitute a simulation, because specific steps of a simulation can hardly be predicted (e.g. dynamic routing paths), but ViNo can be used to approximate the general behavior of a multimedia system. Its strength is the short and flexible expression of transport techniques, which allow simple comparisons on the first sight. E.g., a typical simulation consists of thousands of lines of code. A corresponding ViNo expression consists of one page or less.

The second case study shows the usability of ViNo for complex presentation definitions and for the formulation of search queries. If ViNo is only used for the presentation description, one might argue that existing presentation languages such as SMIL [68] are already integrated in existing players. However, the description length of ViNo is in several orders of magnitude shorter than the same representation in SMIL, which is advantageous for complex presentations such as the tree-like presentation. This can be shown by the following example, where we assume that a user wants to see four video clips (u_1, u_2, u_3, u_4) in parallel. In Listing 4.3 we define the presentation in SMIL. In SMIL one has first to define regions of the window, where the clips will be placed. After that the playback has to be defined and assigned to the regions. If we only concentrate on the playback definition, the code effort is still larger than with ViNo. The ViNo expression for the same representation is $c = [u_1||u_2]||[u_3||u_4]$, where the groups define, which units are represented in one row.

We showed that ViNo is a multipurpose language for short multimedia descriptions by two practical examples. It can be seen that ViNo is a good tool for comprehensive descriptions in research papers and presentations. The definition of the semantics is broad on purpose and we showed that therefore ViNo's usage possibilities are wide spread. However, this broad usage comes with the cost that the application has to specify the semantics itself, e.g., an application has to take care of the QoS negotiation and the wildcard matching.

```
1  <smil xmlns="http://www.w3.org/ns/SMIL">
2  <head>
3    <layout>
4      <root-layout width="640" height="480"/>
5      <region id="ID1" right="55%" bottom="55%"/>
6      <region id="ID2" left="55%" bottom="55%"/>
7      <region id="ID3" right="55%" top=55%"/>
8      <region id="ID4" left="55%" top="55%"/>
9   </layout>
10  </head>
11  <body>
12      <par>
13          <video src="u1.mp4" region = "ID1"/>
14          <video src="u2.mp4" region = "ID2"/>
15          <video src="u3.mp4" region = "ID3"/>
16          <video src="u4.mp4" region = "ID4"/>
17      </par>
18  </body>
19  </smil>
```

Listing 4.3: The presentation of 4 parallel video clips using SMIL

5. Non-sequential Multimedia Caching

In this section the topic caching and replacement is discussed with the focus on videos that are consumed in a different order than it was produced, i.e., *non-sequential media access*. This brings new challenges for multimedia delivery. Caches cannot rely on the sequential pattern of videos anymore, new patterns have to be detected and used for prefetching and replacement. Evaluations on this topic were made in [1] and [6] and are adapted for this chapter. The results are the basics for future self-organizing delivery patterns.

5.1. Related Work

Non-sequential media access is not yet a topic on its own. However, there are similar topics such as branching, prefix caching, segment based caching and jump prediction in video streams that can be related to non-sequential media caching. In the following an overview of these topics is given (see [1], [6]) .

Traditionally, video caching techniques have to be capable of handling large video files. To simplify this task, a lot of research has been done on partial caching [86]. In general the partial caching idea can be mapped to non-sequential media access. Researchers analyzing segment based caching found out that the segments in the beginning are the best to cache. This is shown in [87], where the authors measure the popularity between segments, resulting in an internal popularity distribution. Based on this the authors introduce a caching algorithm for streaming media. Considering fixed sized segments of one second, they observed that the popularity within a video follows a k-transformed Zipf-like distribution (for $k_x = 10$ and $k_y > 200$). Since the internal popularity shows that the beginning of the video is most popular, the caching policy prioritizes prefix caching. Non-sequential media, however, does not have a notion of order. E.g., there is no *beginning* of a collection of units and

therefore prefix caching is not applicable. Non-sequential video caching can be rather regarded as a generalization of prefix caching.

Chen et al. in [88] proposed a more dynamic approach. This caching policy applies each time an object is accessed. If the object is accessed the first time it is fully cached. If the object is not accessed the first time, but fully cached, nothing is done. If an object is not accessed the first time and it is partially cached, then the cache tries to get more segments. The cache admission policies are supported by adaptive replacement strategy that calculates the utility of each object. Then the object with the lowest utility is selected and on this object the segmentation is done. If the object is fully cached, only the first segments are kept and the rest is removed from the cache. If the object is partially cached, the last segment is removed from the cache and the rest is kept.

Another caching mechanism is introduced by Guo et al. in [89]. The authors observed that no caching technology supports interactivity like *jumps* in a video stream. Therefore, two segment caching techniques are introduced: (1) basic interleaved segment caching (BISC) and (2) dynamic interleaved segment caching (DISC). BISC prefetches every second segment. This guarantees a higher hit rate if interactive jumps are common. On a miss, the cache delivers the closest cached segment, because it is more likely that after a jump segments are accessed sequentially. DISC is the extended version of BISC and initially caches a whole object for observation. Later, it will be decided if segments are replaced or several segments are stored sequentially. Both caching policies expect sequential consumption of the videos and therefore only support forward jumps.

Zhao et al. in [90] define *non-linear* media as a video that consists of several parallel branches. The user decides at certain points which branch to follow, leading to different story paths for the same video. The streaming system holds one channel per branch; the content is transmitted using multicast. In contrast to traditional movies, there is a need to predict the paths that will be chosen by the user. This can lead to high startup delays at each decision point. Nevertheless, the authors showed that some hints regarding the client branch selection are enough for server bandwidth and client data overhead reduction. The idea is to collect information about paths chosen by other clients and then prefetch popular segments.

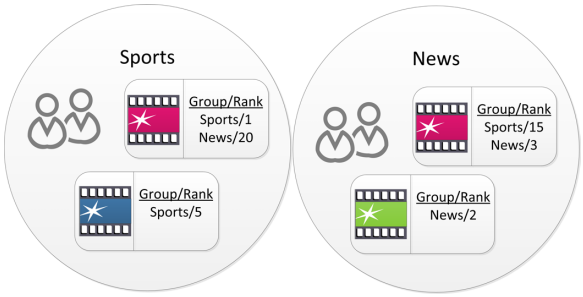


Figure 5.1.: Two semantic groups with distinct users and partly shared content

5.2. Flexible Caching

In the context of the SOMA project (see Chapter 2) flexible caching is of specific interest. Ideally, a caching technique has to map the flexibility of the unit model - i.e., non-sequential media access. Since a user has the possibility to request whatever unit he/she likes, a proactive cache for non-sequential media needs information about the content and the users to be able to identify consumption patterns.

It is assumed that a smart user application exists (see Section 4.4) that allows for request formulation and provides information about user intentions (see [91]). User intentions are described by metadata about semantic roles a user can be categorized to. Such a role could be, e.g., *informational* denoting users looking for many but unspecific data and *transactional* denoting users wishing to buy a specific content. Another role could be the interest in a specific genre. As an example, one group is interested in *sports*, the next into *news* or *art*.

To bring users and content together, the content has to be described by metadata. In this context we define that a unit consists of a video (or a part of a video) and its corresponding metadata. This metadata can be matched to the user's intentions (e.g., by equality). For the sake of simplicity we assume that one user can be described by a single semantic role and only requests units that match this role. However, one unit can be matched to different user roles. User roles form therefore semantic groups which contain users and units. In each group some units are more popular than others, therefore units in a group can be sorted by their popularity ranks (where 0 is the best). An example for two user groups, Sports and News, is shown in Figure 5.1.

The task of a cache is to organize units of different user roles and to identify the most promising units for future use. So, units of different groups compete for the

space available in the cache. It is assumed that a flexible cache knows about existing groups and about the current popularity rank of a unit in each group.

Based on this prerequisites two admission policies are discussed. Both admission policies assume that prefetching of a unit improves the hit rate of the own user group and also supports other user groups, if the taste is overlapping. The first admission policy is based on the idea that units with similar popularity ranks are most likely to be requested within a short period of time. Initially, the cache is filled with the most popular units of each group.

Subsequently, a user with role r_1 requests a matching unit with popularity rank p within r_1. The cache expects the user to request further units with similar popularity and therefore prefetches the unit with the next popularity rank $p+1$ within r_1 ($u_{current} \leftarrow u_{next}$ if using ViNo). If both units are already in the cache, nothing has to be done. Otherwise, one or both units have to be loaded from the origin server.

This policy is called *simple cache admission policy* formally defined as follows:

$$\text{prefetch}=\begin{cases} u_{next} & \text{if hit } u_{current} \\ 0 & \text{if hit } u_{current} \text{ AND hit } u_{next} \\ u_{current} \leftarrow u_{next} & \text{else} \end{cases}$$

This policy does not differ between popular and unpopular units. Thus, if a user requests an unpopular unit the next even more unpopular unit is prefetched as well. Therefore, the idea is to improve the admission policy, by evaluating the *goodness* of a unit. If the requested unit is popular enough among all groups, the next popular unit is worth to be loaded as well. This is done by a rank aggregation of the unit over all groups. If the calculated global rank r_{all} is among the best x units, the next unit is considered for prefetching. The threshold x has to be predefined.

We want further prioritize units with a high popularity in at least one group to units having average popularity in all groups. To ensure this precondition, the logarithm is used. The overall calculation is defined as follows, where r_i defines the rank within group i:

$$r_{all} = \frac{1}{n} \sum_{i=1}^{n} \ln r_i$$

The prefetching of a unit works as follows. If the current unit's r_{all} is smaller than the logarithm of a predefined threshold rank x it is popular enough and the next popular unit can be prefetched. Otherwise only the current unit is loaded. The rank-based prefetching algorithm is defined formally:

$$\text{prefetch}=\begin{cases} u_{next} & \text{if } r_{all_{current}} < ln(x) \text{ AND hit } u_{current} \\ 0 & \text{if hit } u_{current} \\ u_{current} \leftarrow u_{next} & \text{if } r_{all_{current}} < ln(x) \text{ AND miss } u_{current} \\ u_{current} & \text{else} \end{cases}$$

Usually, the cache size is limited and content has to be replaced intelligently. Simple replacement strategies such as least recently used (LRU) proved to be efficient and easy to implement. In the next section the combination of prefetching and replacement are evaluated and discussed.

5.3. Evaluation

We evaluated the proposed algorithms by a simulation, which is based on Omnet++ [85]. The architecture of the simulation consists of a cache, an origin server and a number of user groups connected to the cache. The user requests are generated with Medisyn [92], which models the relationship of popularity and content by using a Zipf-like distribution. We create 100 units with different popularity for each user group. The performance of the policies is compared for different cache sizes; 5, 10, 20 and 40 %, where the cache size of 100 % means that all 100 units can be stored.

In the following the proposed admission policies are evaluated in three scenarios. The first scenario investigates the behavior of a cache if two user groups are connected, which do not share the same interests. The simple admission policy is compared to the rank-based policy. Both are combined with least recently used (LRU) replacement. The second scenario randomly assigns the popularity of a unit for two user groups, which leads to a higher number of units that are popular in both groups. Additionally, the replacement strategy is extended from LRU to rank-based replacement. The third scenario investigates the differences of the admission and replacement policies if more than 2 user groups have to be managed.

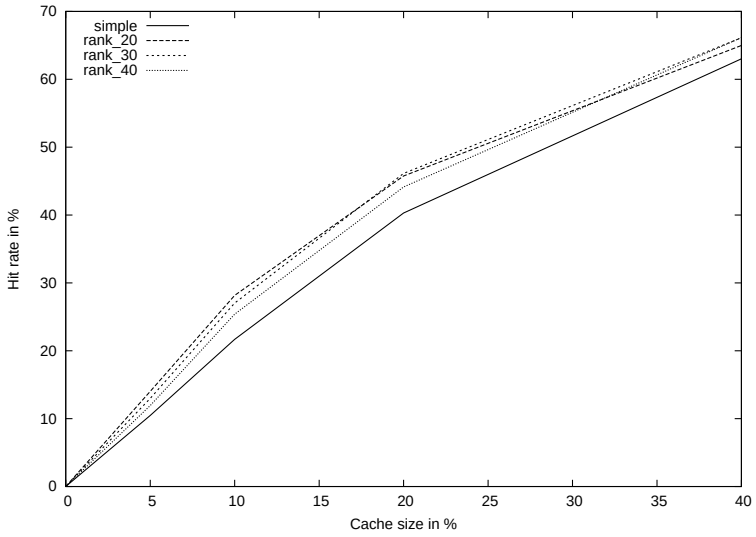


Figure 5.2.: Hit rate comparison of two competing user groups using simple and rank-based admission with LRU [6]

The admission policies are compared by measuring the impact on the user's satisfaction, which can be done by comparing the hit rates. Since the user satisfaction does not come without cost we further evaluate the efficiency of the prefetching mechanisms by measuring the load on the server. The load on the server consists of the forwarded requests on a miss and the prefetching requests.

5.3.1. Scenario 1: Two Competing User Groups

In this worst case scenario the cache has to handle two user groups with contrary interests, because then units interesting for both groups are rare[1]. E.g., one group is interested in sports and the second group in movies.

The rank-based algorithm has to be configured at startup by defining the rank threshold x (see formal definition above). In this scenario, three different thresholds are chosen: ranks 20, 30 and 40.

In Figure 5.2 the hit rates of the different policies are depicted, which are aggregated for all user groups. One can see that both the simple and the rank-based policies are

[1]This section is adapted from [6]

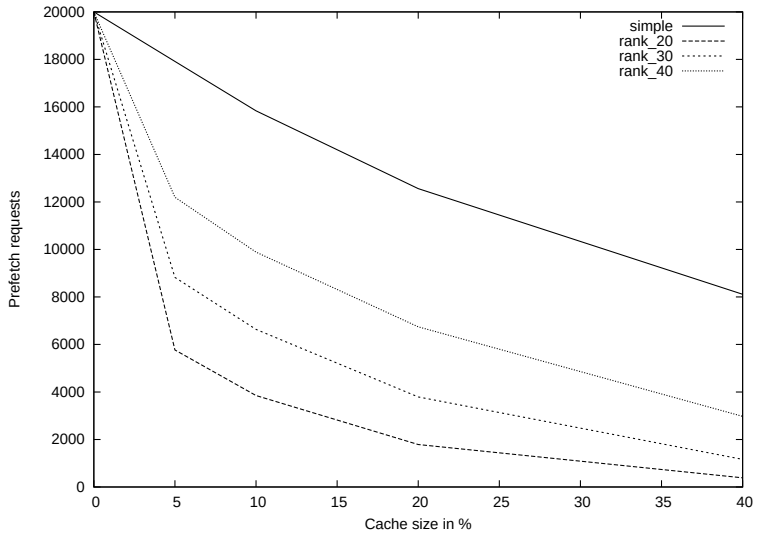


Figure 5.3.: Comparison of prefetching requests of two competing user groups using simple and rank-based admission [6]

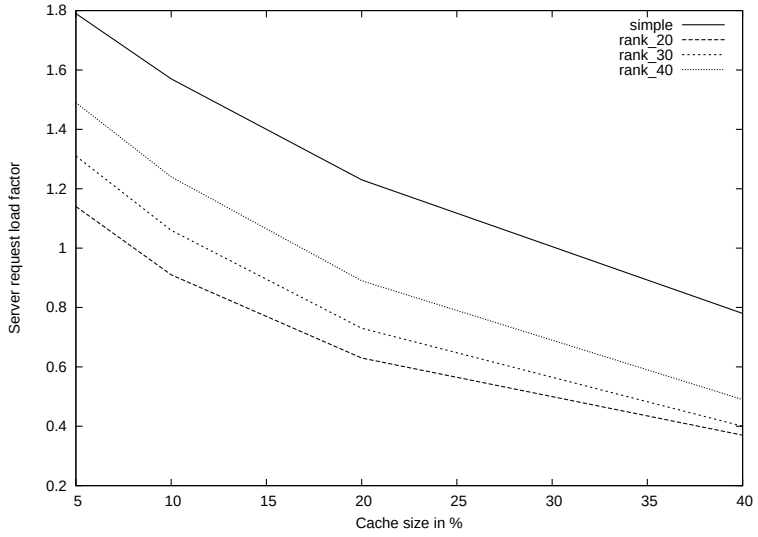


Figure 5.4.: Requests forwarded to the server using simple and rank-based admission with different cache sizes [6]

capable of fulfilling non-sequential requests. However, the simple admission policy leads to a higher number of replacements because of the less restrictive prefetching. The rank-based algorithm has in average a 5 % higher hit rate than the simple admission policy for all cache sizes. Since the rank-based algorithm is stricter regarding prefetching requests, the units remain longer in the cache.

Although the hit rates are similar, the difference in performance can be seen in Figure 5.3, where the number of prefetches is depicted. The rank-based admission policy is more efficient by reducing the prefetches remarkably. Thus, for such scenarios a threshold of 20 is recommended. Furthermore, the cache size should be larger than 5 % to reduce the number of replacements.

This is also seen at the request fulfillment efforts of the origin server, which are depicted in Figure 5.4. The rate is 1 if each client request is directly fulfilled by the server (i.e., in this scenario 20,000 requests go to the server). If the rate is higher than 1, the prefetching increases the server load. Thus, prefetching is of no use. So, the admission policies start to be efficient if the rate of requests to the server is below 1.

The simple admission policy reaches this point at a cache size of 30 %, whereas the rank-based policy reaches this point already at a cache size of 10-15 %. The main reason for additional requests to the origin server is the replacement of the wrong units. Although LRU replacement supports popular units, a smarter replacement strategy is needed.

5.3.2. Scenario 2: Two Groups, Two Replacement Strategies

In this scenario each user group selects a number of random units of their interest, which led to a higher number of units that are popular in both groups[2]. As a consequence we configured other thresholds (x) for the rank-based algorithm, i.e., ranks 10, 15 and 20.

We further compare the admission policies to pure LRU replacement without prefetching to show the differences to a traditional cache.

The results of the hit rate comparison are depicted in Figure 5.5. It is shown that the

[2]This section is adapted from [1]

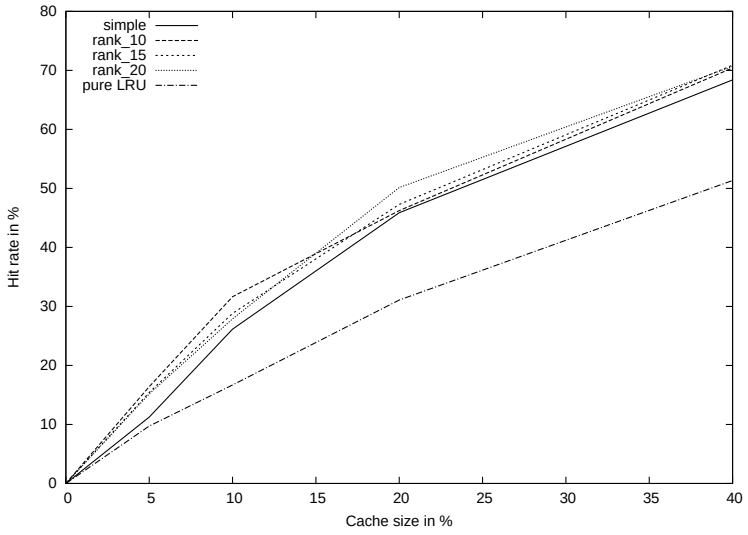


Figure 5.5.: Hit rate comparison of pure, simple and rank-based admission using LRU for two user groups with overlapping interests [1]

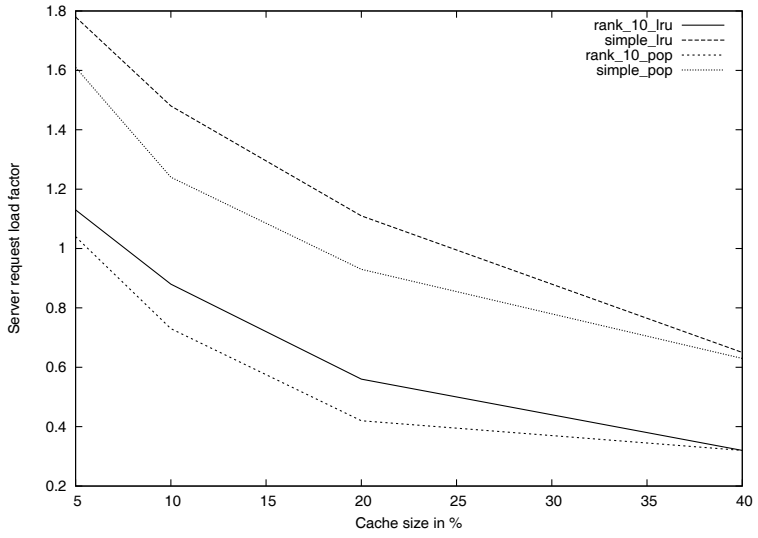


Figure 5.6.: Factor of server requests compared to user requests (LRU vs. popularity replacement) [1]

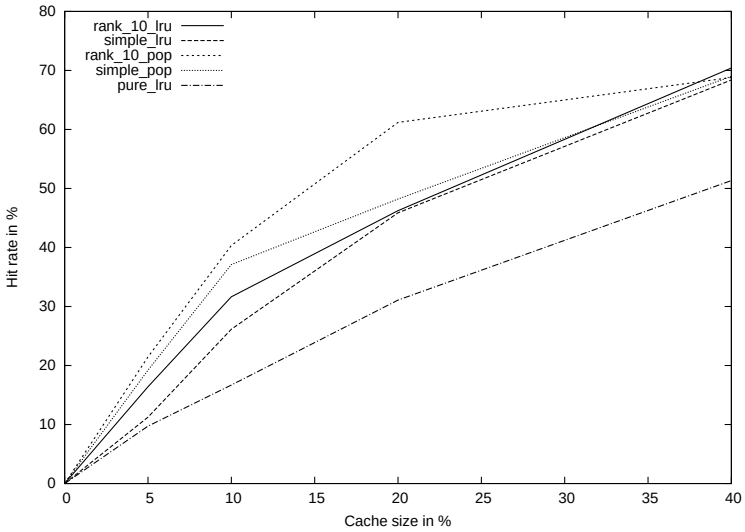


Figure 5.7.: Hit rate comparison of the admission policies using LRU and popularity replacement [1]

rank-based algorithm behaves similar to the competitive user groups before. However, since the interests of the users overlap, the rank-based algorithm works more efficient. Additionally, in comparison to pure LRU the improvement is up to 20 %. It can be seen that also in this scenario the thresholds of the rank-based policy show small differences in hit rate, but the server load differences are large. Therefore, further experiments are limited to a threshold of 10.

As discussed in scenario 1, a high number of unnecessary replacements are done if using LRU. The idea is to apply a popularity based replacement. Therefore, the application of the rank-based policy for replacement is discussed in the following.

Figure 5.6 depicts that the popularity based replacement outperforms LRU for both the simple and rank-based admission. The cache size for the simple admission can be smaller by two thirds. The rank-based admission seems to perfectly profit from the popularity based replacement and needs only a basis cache size of 7 %. However, one might argue that the rank-based admission in combination with rank-based replacement might be too complex for caches with limited computing resources. For these cases the simple admission policy in combination with rank-based replacement is preferable.

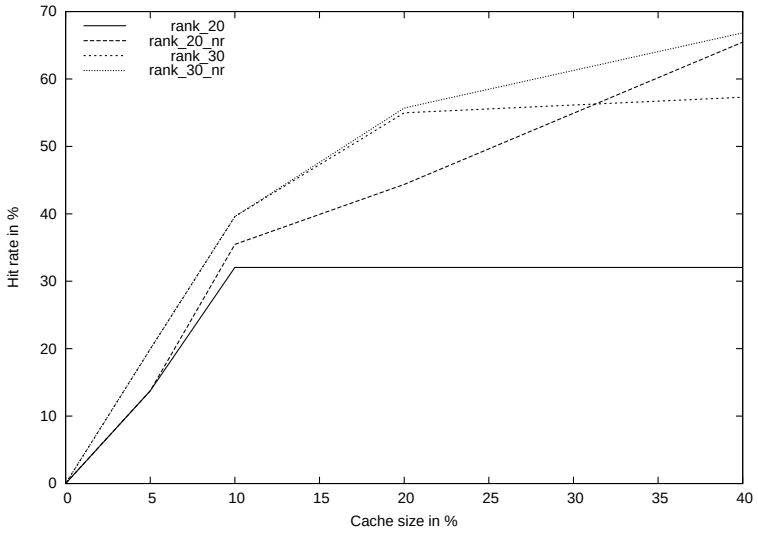


Figure 5.8.: Hit rate differences with restrictive and non-restrictive cache for 4 user groups

Figure 5.7 compares the hit rate evolution for both replacement policies. The comparison to pure LRU shows that the hit rate increases remarkably if applying a combination of rank-based admission and rank-based replacement. For a cache size of 20 % the hit rate differs by 40 %.

5.3.3. Scenario 3: Four Groups

In this scenario the differences in the caching behavior are investigated if 4 user groups are connected. Each of the user groups sends 10,000 requests. We concentrate on the behavior of the rank-based admission with rank-based replacement, whereas the rank-based replacement is restrictive against units to be cached. This means that a unit that is not popular enough can never be cached.

In this context we observed interesting behavior of the rank-based policy as shown in the hit rate comparison in Figure 5.8. If we use a threshold of rank 20, the performance remains steady although the cache size is increased. It shows that the caching policy leads to unused cache space. The same can be seen in the case of the threshold of 30, but later. Therefore, an extension of the caching policy is needed,

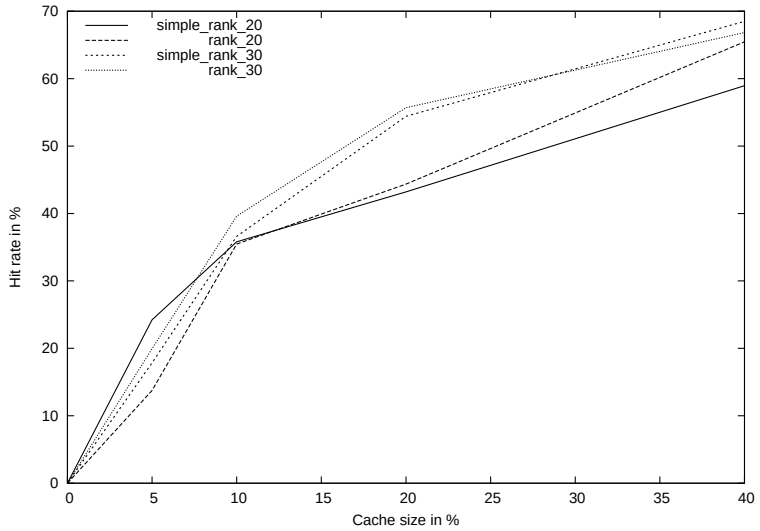


Figure 5.9.: Hit rate comparison for 4 user groups

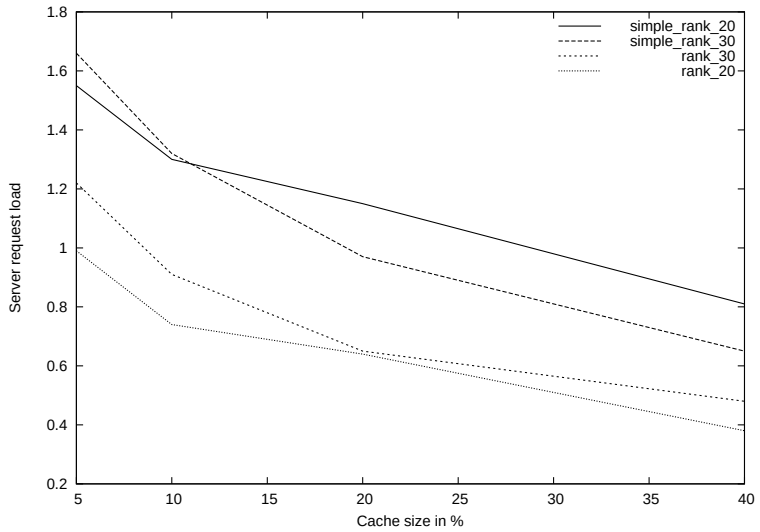


Figure 5.10.: Server request load comparison for 4 user groups

which allows every unit to be cached if there is enough space. If the cache is full and a unit arrives with a good rank, replacement is done. The differences regarding the hit rate are remarkable. In all future graphs we reduce our evaluations to the non-restrictive cache.

In the following we further compare the rank-based algorithm to the simple algorithm, both with rank-based replacement. Figure 5.9 shows stable results for the rank-based algorithm even with more user groups. However, the simple algorithm shows better hit rates if the cache is small, and similar if the cache size is about 10 %. Rank_30 is outperformed by rank_20 at large cache sizes, which shows that rank20 has a better tradeoff between restrictiveness and user satisfaction.

The rate of requests forwarded to the server in comparison to the rate of user requests is depicted in Figure 5.10. It can be seen that simple_rank_20 never brings the server request rate below 1, thus is never efficient. simple_rank_30 needs at least a cache size of 20 % for reducing the server load. In comparison to that the rank_20 algorithm is efficient from the start, and rank_30 needs a cache size of 10 %.

This last scenario shows that the rank-based algorithm is capable of several user groups. However, it can be seen that the threshold has to be adaptive, because in most cases the number of user groups is not known at system startup. Additionally, users might change their interests over time.

5.4. Summary and Discussion

A proactive non-sequential cache has been introduced, which is based on the idea that user intentions can be semantically categorized. This information is used to predict user behavior by mapping the units to user groups. The prediction is an aggregation of the popularity ranks for one unit within all groups. It evaluates the goodness of a unit for being stored in the cache and for prefetching the next unit regarding the popularity rank.

It is shown that a popularity based evaluation of a unit helps predicting the right units for future requests. Thus, it is possible to predict request patterns of units that are likely to be consumed together. It is noteworthy that this caching technique does not need the notion of order, thus is applicable for non-sequential media access.

However, if being too restrictive the efficiency drops, thus, the storage of the cache has to be open for less popular units.

Whereas the applicability for non-sequential media is shown in this chapter, the applicability of self-organizing systems has to be investigated in the next chapters. Since self-organizing systems rely on local information rather than on global information such as popularity, the intention categorization has to be changed.

6. Bio-inspired Self-Organizing Multimedia Delivery

In this chapter the actual delivery of non-sequential media is discussed. A complex environment at social events is expected and it will be shown that self-organizing algorithms can cope with this complexity. Advantages and challenges are discussed as well as the applicability of bio-inspired algorithms for multimedia transport investigated. In the context of this chapter a unit comprises continuous data and metadata; e.g., video plus audio plus metadata, but also photos, because they are presented for a specific duration (e.g. 3 seconds). The focus is set on multimedia search and replication in unstructured dynamic networks. This chapter is adapted from [2], [7] and [8].

6.1. Introduction and Related Work

As described in Chapter 3 many researchers design their technical self-organizing algorithms by adapting principles found in nature. We propose a self-organizing algorithm, which is inspired by the human endocrine system. In the endocrine system glands such as the epiphysis create hormones. The hormones are released to the blood system over which they travel to target cells, where they lead to specific actions. The actions depend on the type of the target cell and can be different although the hormone is the same. The hormone level of the body is typically influenced by positive and negative feedback. For example, the epiphysis creates melatonin that regulates rhythmic behavior such as the sleep-wake cycle. During the night melatonin is released and dock to specific cells in the brain. Additionally, during winter more melatonin is released because of the number of dark hours. So, the positive feedback (darkness) amplifies the creation of melatonin, whereas the negative feedback (daylight) stops it [93].

These principles are already adopted by researchers, who build artificial hormone-systems. For example, Brinkschulte et al. in [94] implemented an artificial hormone-system for task allocation in grids. The nodes in the grid compete for the execution of different tasks, where the best node has to be found. Each node, interested in executing a task spreads artificial hormones to agree on the best node. The goodness of a node is defined by its *eager value*, which is influenced by the nodes of the system that spread *accelerators* and *suppressors*. Both values are added and subtracted from the eager value, thus, represent positive and negative feedback. The communication of the nodes is done by broadcast messages. The node with the highest eager value can execute the task and sends suppressors to all other nodes to prevent duplicate execution.

The described artificial hormone system is interesting because it supports several types of hormones, each for a different task. A disadvantage, however, is the communication effort caused by the usage of broadcast messages. The SemAnt algorithm [64] as discussed by Michlmayr (see Chapter 3) is more efficient. SemAnt is an ant based algorithm, where only a limited number of ants travel around the network. The proposed algorithms in this chapter strive to cover the advantages of both SemAnt and artificial hormone system.

The underlying system of the proposed algorithm is an unstructured peer-to-peer overlay. On the one hand we want to ensure content availability and on the other hand we want to reduce the search space. For this reason the focus is set on smart replica placement in the dynamic environment. The number of replicas and their location has to be adaptive. For example, we could use the replication strategy as proposed by Rong in [95]. The number of replicas depends on their utilization rate. Temporarily popular videos are replicated more often and replicas are destroyed when the number of requests for the videos decreases. Rong solves the management of utilization rates by introducing a central entity. An example for distributed replica placement is discussed by Herrmann in [96], where the cost of a request is measured in number of message transmissions per time unit to serve client requests. The goal is to minimize these costs without global coordination. Regarding the number of replicas the author defines a parameter ρ that describes the coverage radius of clients. If the number of requests for this service is below a given threshold, the service replica removes itself from the system. The challenge is to define a global cost function for the replication.

To ensure efficient replication we rely on *cooperation*. Cooperation is highly discussed in the science of evolution [97]. In evolution the fittest individuals survive and can generate more offspring. So, each individual will take care of (1) increasing its own fitness (2) not to cooperate to increase other individuals' fitness, thus being *selfish*. The opposite of pure selfishness is *ultrasocialism* where examples are the human society and specific species of ants, bees, etc. In these examples the individuals cooperate and even sacrifice themselves for the others leading to an increased performance of their environment. E.g., a group of wolves can hunt larger deer than a single one by being cooperative.

Cooperative behavior is a research topic in peer-to-peer systems as well. Research has been done to introduce incentive systems to increase the cooperation of peers and to reduce the number of freeriders. *CrossFlux* [32](described in Chapter 3) is an example for a peer-to-peer architecture for multimedia streaming, which concentrates on the improvement of cooperative behavior. A node that provides much upload bandwidth should get better quality videos in return. If nodes do not cooperate the whole system performance can degrade. In our system a node that cooperates by providing storage and bandwidth has a better chance to get better service quality in the future. I.e., on the transport path a node has the chance to decide if the current unit might be relevant for it in the future. In a real scenario such as the Ironman privacy issues might cause people not to accept the application. This could be solved by introducing the notion of friendship as used by social web networks when building the overlay. Another possibility would be to introduce cloudlets as described by Satyanarayanan (Satya) in [98]. Cloudlets are proxies that can be accessed at the social event and be used by the visitors as storage facility. The interesting part of these cloudlets is their mobility, i.e., they can be moved around the event to the most needed places with reporters or organizers.

Beyond these discussions in the rest of the work it is assumed that the overlay construction is already finished and that the peers provide shared storage for multimedia transport. The proposed algorithm is, however, capable of managing peer churn.

6.2. Algorithm Description

The idea is that units place and replicate themselves where they are needed at the moment or in the future (see [2], [3], [7] and [99]). Patterns of units often consumed

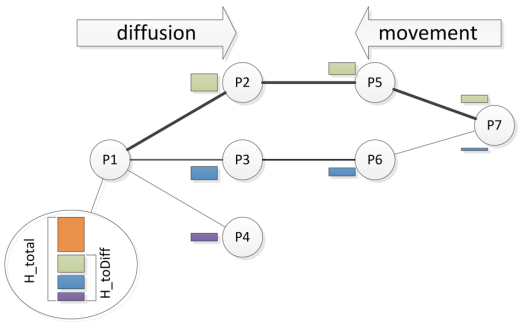


Figure 6.1.: Example for a hormone trail for a unit. Thick connections mean better QoS [7]

together will emerge. A network of nodes and units is seen as an artificial hormone-system where the nodes are the cells creating and consuming hormones and the links between the nodes represent the blood stream. Hormones indicate interest in a specific unit or in its content. Hormones are spread to the network in such a way that a hormone trail evolves between the requester and the resource holding node, where the highest hormone concentration is found at the requester. Thus, a unit is transported by following the path with the highest hormone concentration until the requester is reached. The hormone concentration is balanced by positive and negative feedback. Positive feedback is reached by creating further hormones and then to release them to the system. Negative feedback is reached by (1) evaporation to remove alternative paths and (2) deletion of hormones if a corresponding unit is stored on the node.

The following example shown in Figure 6.1 depicts the algorithm principles. The physical units are spread over an unstructured network of nodes and the task is to bring the units to the requester. Given that $P1$ requests a unit u, which is located at $P7$. $P1$ creates hormones and diffuses a part of them to its neighbors (H_{toDiff}), the strength is dependent on the QoS provided by the corresponding link. The neighbors forward also a part of their hormones, until a node with the requested unit is found or no hormones are left for forwarding. In this example the unit is found on $P7$ and it will be moved over the strongest path over $P5$ and $P2$ to $P1$. Note that in the case of node failure the unit can take alternative paths.

The algorithm is cyclic and each peer in the network has to handle the tasks as shown in Algorithm 1. The tasks are executed until a specified time stamp max, which could be the end of a social event.

Algorithm 1 Execution loop of each peer

1: **repeat**
2: handle incoming requests
3: diffuse hormones
4: move units
5: evaporate known hormones
6: $timestamp \leftarrow timestamp + 1$
7: **until** $timestamp = max$

At the execution of the first task peers accept different request types: sequential, parallel and mixed. Sequential requests contain a number of units with a given order. The presentation starts if the first unit is available. Parallel requests contain a number of units without any order, however, requires that all units are available before the presentation, because all units start at the same time. Mixed requests consist of arbitrarily nested sequential and parallel parts, and have to be parsed to define the order of presentation and the requirements to the stored units.

The basic principles of handling incoming requests are shown by the example of parallel requests (see Algorithm 2). If a unit is not present, the corresponding hormone H_{u_i} is increased by η_0 in the case of a new request and by η otherwise (all parameters are explained in Section 6.3). If all units are found locally, the presentation starts. The incoming sequential request is handled similarly, but it also takes playback time and the order of the units into consideration. Units closer to the playback time get more hormones than the later units, for implicitly increasing the importance of those units.

The diffusion of hormones is performed as shown in Algorithm 3. If a unit for the corresponding hormone is stored on the current node, the demand for this unit is obviously fulfilled and the hormones can be deleted. This avoids the attraction of further replicas of this unit. If the required unit is not stored on the node a part of the corresponding hormone is subtracted. We call the subtracted part $H_{i_{Diff}}$, which is further distributed among the neighbors according to their QoS weight w. To reduce the number of messages, the hormones to forward can be collected in a queue before sending.

The next step is to take care of the guidance of units, which is shown in Algorithm 4. A unit will be moved to the neighbor with the highest hormone value max_{H_i} for

Algorithm 2 Handle incoming parallel request

Require: generated request

for all u_i in request **do**
  if u_i is stored locally **then**
    count $\leftarrow$ count + 1
  else
    if $request = new$ **then**
      $H_{u_i} \leftarrow H_{u_i} + \eta_0$
    else
      $H_{u_i} \leftarrow H_{u_i} + \eta$
    end if
  end if
end for
if count = size of request **then**
  present all units
end if

Algorithm 3 Diffuse hormones to neighbors

for all H_i in stored hormones **do**
  if u_i for H_i is stored locally **then**
    $H_i \leftarrow 0$
  else
    $H_{i_{Diff}} \leftarrow H_i * \alpha$
    $H_i \leftarrow H_i - H_{i_{Diff}}$
    for all N_j in neighbors **do**
      $H_{i_{N_j}} \leftarrow H_{i_{Diff}} * w$
      forward $H_{i_{N_j}}$
    end for
  end if
end for

the unit. To migrate a unit to this neighbor, max_{H_i} has to be larger than the local hormone plus a migration threshold m. In the initial version of our algorithm a simple replication mechanism is introduced. If a unit is currently in presentation it will be copied, otherwise moved. In the later chapters different replication mechanisms are evaluated. Before transport, units are collected in output queues. Each node has one output queue per neighbor. These output queues are sorted, i.e., a unit with higher hormone concentration is delivered first.

Algorithm 4 Move units

```
1: for all u_i in storage do
2:   if u_i in presentation then
3:     copy ← true
4:   end if
5:   get maximum max_{H_i} hormone from neighbors
6:   if max_{H_i} > H_i + m then
7:     if copy then
8:       copy u_i to output queue towards this neighbor
9:     else
10:      move u_i to output queue towards this neighbor
11:    end if
12:  end if
13: end for
```

The final step is the evaporation of hormones, which is described in Algorithm 5. The evaporation has two tasks, first reducing all known hormones by ϵ and second to delete all hormones with a value below the threshold t.

Algorithm 5 Evaporate known hormones

```
1: for all H_i in stored hormones do
2:   H_i ← H_i − ε
3:   if H_i ≤ t then
4:     delete H_i
5:   end if
6: end for
```

At the beginning the location of the units may be random, but at some point the location of the unit converges to the right places. Another important fact is that only requested units move around and replicate themselves, the unpopular units stay where they are. The overall placement and delivery is done without global control, just by simple tasks performed by each node.

Table 6.1.: Parameters to configure at system startup

ID	Explanation
η_0	Hormone strength for a unit at new request
η	Increase of hormone after each time step by the requester
α	Percentage of hormones to be forwarded to the neighbors
ϵ	Hormone evaporation value
t	Minimum hormone strength
m	Minimum hormone difference to move unit (migration threshold)

6.3. Parameter Settings

The proposed algorithm is self-organizing, however, needs some configuration. In Table 6.1 the necessary parameters are shown. These parameters are dependent on each other; e.g., if the created hormone concentration, represented by η_0 and η, is low and the evaporation value ϵ is high, the movement of units can be limited. The more hormones are created and forwarded, the more hops the hormones can travel and therefore increase the search space. The migration threshold m controls the mobility of units. If m is high, the units need a higher hormone concentration to move to a neighbor, leading to a longer waiting time for the requester. This means that the higher m, the less is the reachable mobility. The parameter settings are essential for the algorithm to work. Therefore, we decided not to tweak the parameters manually, but optimize them by using a genetic algorithm.

As already mentioned in Chapter 3 genetic algorithms are often customized regarding the selection, mutation and crossover approach. In this work a genetic algorithm such as described by Elmenreich in [100] is used, which adapts the selection process.

Initially, the algorithm creates a random population of parameters. Then, it uses elite selection for building the next generations. The candidates are sorted according to their fitness and the best x candidates are chosen. These candidates propagate to the next generation. To reach the same population size as the last generation, the rest of the slots are reserved for mutation, crossover and new randomly generated candidates. For mutation and crossover random elite candidates are chosen.

The fitness function targets client satisfaction, therefore, it optimizes the number of consumed units. The genetic algorithm is integrated into the simulator, which is implemented for evaluating the hormone-based algorithm. For evaluating the fitness

of a parameter set, the simulation is started with this parameter set for a number of runs and the results are averaged. The parameter sets of one population are compared according to their fitness and the result of one generation is the parameter set with the highest fitness. The higher the number of generations the higher is the fitness of the resulting parameter set. The resulting parameter set can be used for all simulations and real implementations of the algorithm for which the system's configuration (e.g., number of nodes, replication type, etc.) is similar to the input of the genetic algorithm.

6.4. Application in a Proxy Network

In this section we evaluate the applicability of the hormone-based delivery algorithm in a proxy network [1]. For this evaluation we implemented a cyclic simulator that allows for different client request models, network models and provides statistical support [2]. In the following the settings for this scenario are described.

Each proxy performs the tasks described in Algorithm 1. Additionally, it controls its storage by a clean-up mechanism called *smart clean-up*. If the storage space at a proxy reaches a certain limit, it tries to get rid of some units by deleting them or by moving them to other nodes. A unit can be deleted only if there is a copy of it in the neighborhood. This restriction is based on the requirements of the first use case described in Chapter 2, which avoids that units vanish from the system. If the unit cannot be deleted, the proxy tries to move it to a neighbor; e.g., if a neighbor has a higher hormone strength, or more storage space.

We assume that a unit consists of data and metadata. The metadata contains a content description, which we model as a three dimensional array of integers (in other settings the array might have any dimension). Each integer identifier describes a predefined tag. E.g., a unit can be described by swim=1, people=5, loud=45, which leads to a content array [1,5,45]. Assuming that similar identifiers describe similar tags, one can compare two content arrays by applying the Euclidean Distance.

Each client starts with a random content array, which describes its taste. Based on this taste the client requests units with similar content arrays. After watching

[1] This section is adapted from [2].

[2] The simulator used in this section is published under GNU GPL at http://code.google.com/p/videonetwork/.

η_0	η	α	ϵ	m	t	c
0.40	2.48	0.26	0.08	0.09	0	31

Table 6.2.: Parameter settings of the proxy scenario [2]

a unit the taste of the client might change, e.g., "I want to see more like this". In this scenario the clients are aware of unit IDs and therefore one request consists of a sequence of unit IDs. A new request is generated only if the current one is fulfilled.

The hormone-based search and delivery will be compared to a reference algorithm. The reference algorithm uses iterative deepening and then delivers a unit hop-by-hop according the shortest path found. We further compare LRU with the smart clean-up strategy.

6.4.1. Evaluation Settings

We generated the input parameters for the simulator by using the genetic algorithm described in Section 6.3. The fitness function is chosen to maximize the number of consumed units. In Table 6.2 it can be seen that the initial hormone creation parameter η_0 is low in comparison to the periodical increment parameter η. Thus, the hormone concentration increases over time, leading to an increasing search space. The value ϵ for the evaporation on the alternative paths is very low. The low migration threshold m increases the probability of unit movement. The minimum value for a hormone before deletion t is set to zero, which means that hormones remain long on the nodes. The final parameter c describes the storage percentage from which the clean-up procedures are triggered. Its low value further increases unit movement. So, with this parameter set the support for the movement of distant units is the main goal.

In this scenario we evaluate different network sizes with 5 , 10, 20 and 50 nodes. Each node is a proxy that can handle 1 to 20 clients. We implemented the network architecture as connected Erdős-Rényi random graph with a link speed of 10Mbit/s.

The size of a unit is between 50 KB and 5 MB, with an average of 500KB and a standard deviation of 500 KB. Thus, the system allows photos and videos. A request is defined as a sequence of 1-4 units.

At the start of the simulation units are generated until the storage of 10 % for each

node is reached, i.e., the number of units is proportional to the number of proxies. We assume that after the bootstrapping only replicas of the existing units are added to the system. We further limit the storage of one proxy to 50 MB for two reasons, (1) we want to bring the system to its edges and (2) in future scenarios where mobile phones should act as peers the storage will be limited as well. An acceptance study of the application, which would also include such parameters, is part of future work.

We average the results for every experiment using 10 different runs. The simulation runtime is set to 500 seconds.

6.4.2. Results

In this section the algorithm is compared to the reference algorithm in two scenarios: (1) without node failure (2) with node failure. Since the reference algorithm routes the units on the shortest path regarding hops, it is further referred to as *routing algorithm*.

We evaluate the delay, which is measured as the difference between the time of the playback of the most recently unit and the time the current unit is available at the proxy. The delay of the first unit (start-up delay) of the request is measured as the difference between the request time and the startup time. The hit rate is defined as the rate of units, which are immediately available at the target proxy when playback starts (i.e., delay is 0 seconds).

Without Node Failure

In this scenario we investigate if a hormone based system performs equally or better to the routing algorithm regarding delay and hit rate without node failure. Additionally, we compare LRU to smart clean-up.

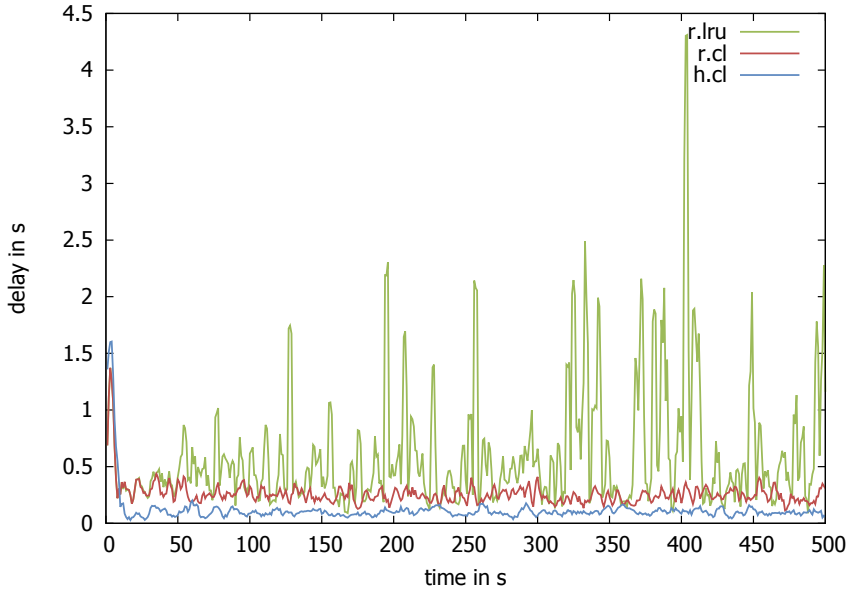


Figure 6.2.: Delay comparison of hormone and routing system for 5 nodes with different clean-up functions [2]

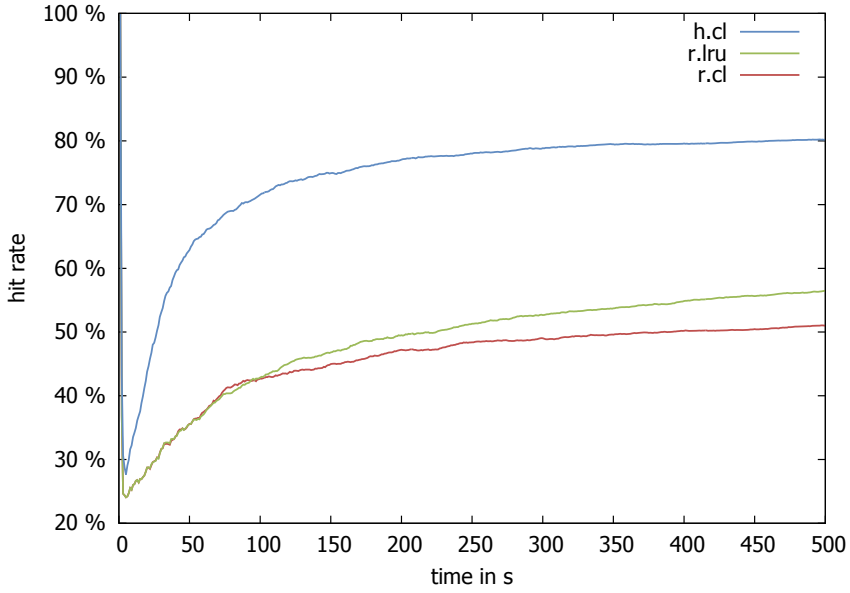


Figure 6.3.: Hit rate comparison of hormone and routing system for 5 nodes [2]

Network of 5 Nodes

Figure 6.2 shows the delay comparison of the hormone-based algorithm (h) and the routing algorithm (r). The routing algorithm is combined with LRU and then with the smart clean-up (r.lru and r.cl). In any case the hormone algorithm (h.cl) outperforms the routing model in combination with smart clean-up and LRU. The main reason for the delay difference is that the hormone algorithm exploits the knowledge about hormones and transports units with higher hormone values first. The delay jitter of r.lru is remarkably high in comparison to r.cl. The reason is that with LRU the replacement of units is more often applied than with the smart clean-up. The restrictive clean-up threshold requires LRU to delete also units that are more popular, because they are more likely to be also available on the neighbors. This results in long travel paths for popular units and therefore high delays.

Lower delay values mean that more units can be consumed and therefore more units are placed correctly. The quality of the placement is measured by the hit rate. Figure 6.3 gives an overview of the results. Although the delay of h.cl and r.cl is similar, the hit rate shows the differences. Hit rates up to 80 % are possible if applying the hormone-based algorithm. It is further interesting to see that r.lru has a 5 % higher hit rate than r.cl in the end of the simulation. Both clean-up mechanisms remove the most recently used units, but the smart clean-up moves the other units to the neighborhood. This storage load balancing leads on the one hand to fewer deletions, but also increases the hop distance.

Network of 10, 20, 50 Nodes

We implemented the 5 nodes scenario to show that hormone-based delivery is possible. However, 5 proxies might not be capable of handling a high number of users. Therefore, we further show the delay development for the hormone model (h.cl) if increasing to 10, 20 and 50 nodes. The delay increases with the number of nodes, since also the number of users and the number of units increase. In Figure 6.4, it is shown that in all of the runs the delay stabilizes at around 150 simulated seconds.

Figure 6.5 depicts the delay development of the routing algorithm (r.cl). The delay is always a bit higher than in the hormone case. E.g., the delay for 10 nodes of the hormone model h.cl is the same as the delay for 5 nodes of the routing model. In all of the runs the hormone system performs better than the routing system. However,

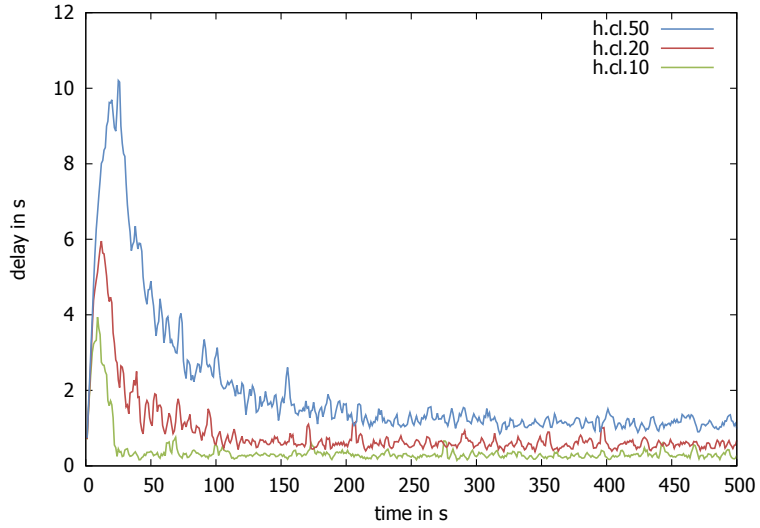


Figure 6.4.: Delay comparison of hormone system for 10, 20, and 50 nodes [2]

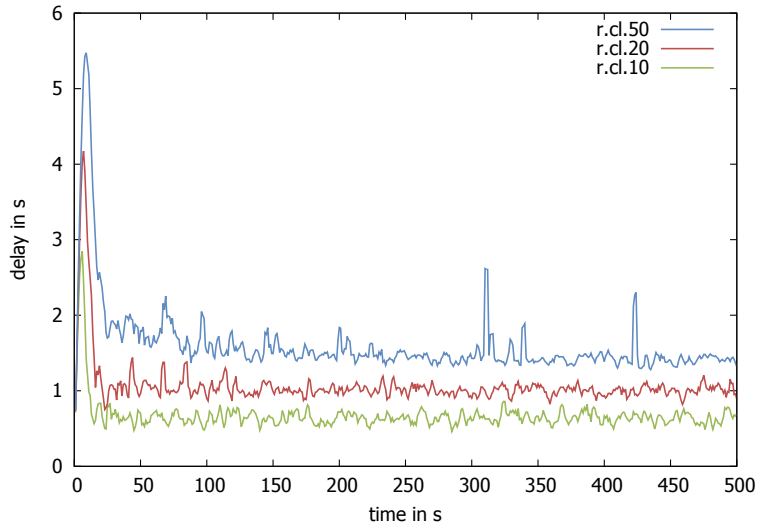


Figure 6.5.: Delay comparison of the routing algorithm with smart clean-up for a 10, a 20, and a 50 nodes network

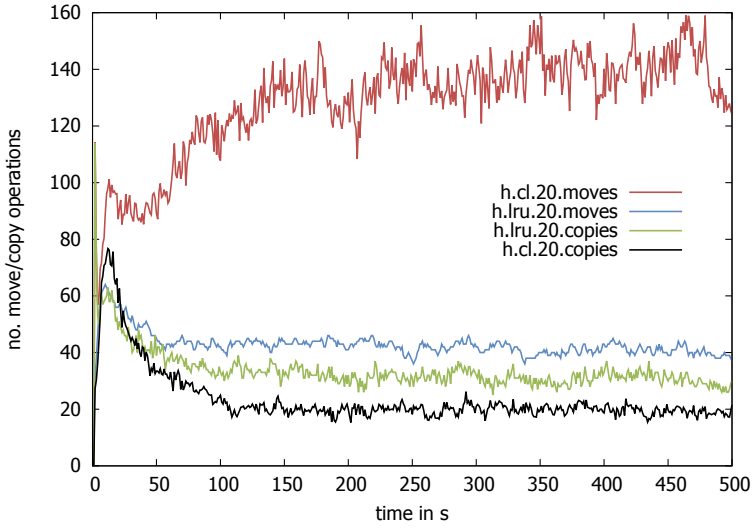


Figure 6.6.: Copy and move comparison if LRU or the proposed clean-up function is applied [2]

during the startup phase the routing algorithm has a lower delay, because it causes fewer movements. In the 50 nodes scenario r.cl has a number of delay peaks, because units near to the playback time are not prioritized such as the hormone algorithm does and therefore some of the units are blocked during transport.

The increasing node number leads to decreasing hit rates, because users have different tastes and compete for low storage capacities. In the 50 nodes scenario the hit rate stabilizes at around 60 % if applying h.cl, but r.cl only reaches 12 %. In the 20 nodes scenario the hit rate of h.cl is 65 % and 16 % of r.cl. In comparison to the 5 nodes scenario the hit rate of r.cl reduces from 50 % to 20 % if the network consists of 10 nodes. Thus, the routing algorithm is less scalable than the hormone algorithm.

Although the LRU clean-up has a bad impact on delay and hit rate, our proposed clean-up function also has its flaws. This is shown in the following by the example of a 20 nodes network. In Figure 6.6 we show that LRU balances the copy and move operations to about the same amount. In the proposed clean-up scheme most of the operations are movements and a very low number of copy operations are done. If we sum up the number of copies and moves the proposed clean-up is more expensive.

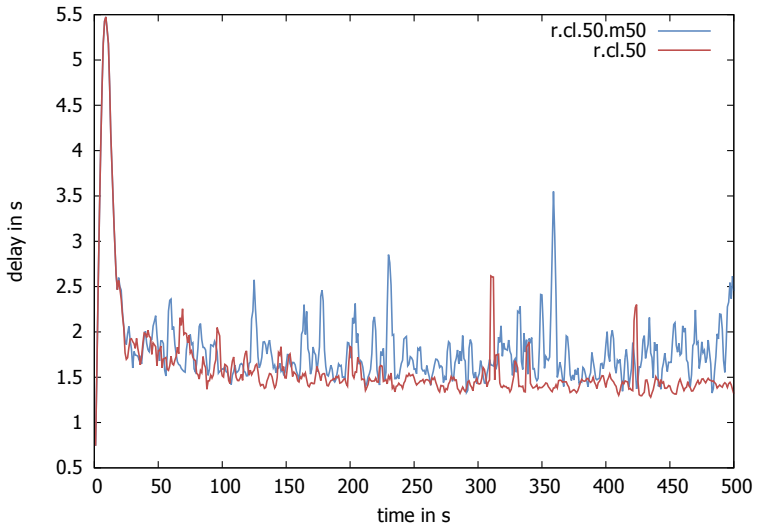


Figure 6.7.: Delay distribution with and without 50 % node failure, routing algorithm, 50 nodes [2]

With Node Failure

In the following we compare the robustness of the hormone model and the routing model regarding the presence of node failures. Since the proxies might be mobile, such as the cloudlets [98], they are also likely to fail. To see how many failed nodes the algorithms can compensate, we periodically remove randomly chosen nodes from the network.

We simulated for 10, 20 and 50 nodes node failure rates of 10 %, 20 % and 50 % (m10, m20, m50). The first two failure modes showed marginal delay differences of around 10 milliseconds for both the routing and the hormone algorithm. In Figure 6.7 the delay differences for the routing algorithm are shown if 50 % of the nodes fail. The routing algorithm has to calculate a new shortest path if any intermediate node fails and therefore causes the delay peaks as shown in Figure 6.7. In comparison to this in Figure 6.8 one can see that when using the hormone algorithm, only slight delay differences are experienced in the end of the simulation. The hormone algorithm does not need any further action on node failures, because alternative hormone trails exist already at request time. Both algorithms can still handle 50 % of node failure, but the performance starts to drop. A higher failure might lead to requests that cannot

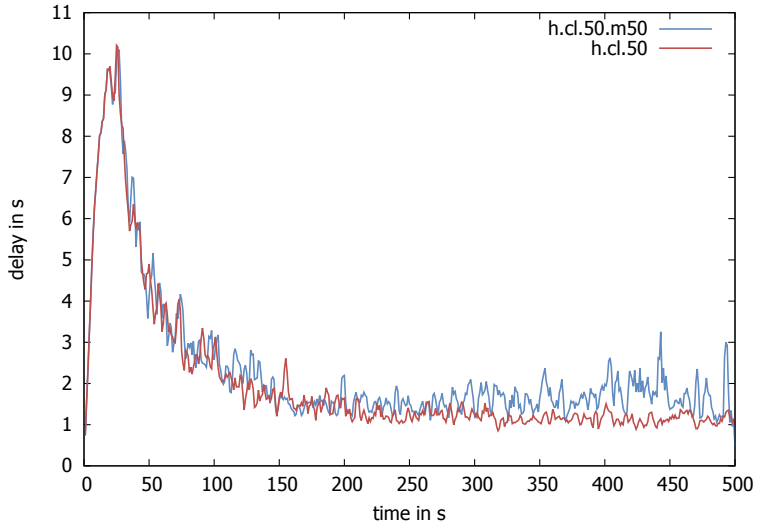


Figure 6.8.: Delay distribution with and without 50 % node failure, hormone algorithm, 50 nodes [2]

be fulfilled anymore, because it is more likely that units vanish from the system.

6.4.3. Discussion

In this section it has been shown that the hormone-based algorithm performs better than the compared routing algorithm. The algorithm is robust against node failure, even if the network size increases.

We considered a moderate number of clients, each connected to one of the proxies. If situations force the appearance of flash-crowds, a peer-to-peer architecture can be advantageous. A peer-to-peer network adapts the architecture of the network to the architecture of the clients and the hormone algorithm has the potential of being used in a network with a large number of nodes. However, the smart clean-up mechanism causes a high number of movements, which is disadvantageous if only one client has to be served per node. Additionally, the proxy network does not exploit the transport path of a unit for replication. In this scenario a unit is only replicated if it is currently in presentation. In the next sections more sophisticated replication and clean-up mechanisms are discussed.

6.5. Replication Strategies for Bio-inspired Delivery in Peer-to-Peer Networks

The application of the hormone-based algorithm to peer-to-peer networks brings new challenges for the placement of units. In this section we investigate different replication strategies to reduce the search space and to increase the robustness for a high number of peers. In an ideal situation the units are already at a peer before they are requested [3].

6.5.1. Existing Replication Strategies

Since we are targeting replication in unstructured peer-to-peer networks we first investigate existing replication strategies. The following categorization is based on [16] and [101].

Owner Replication

The content is replicated at the requester's node [22] and is also called passive replication. Typically, this replication technique is used in file sharing systems based on BitTorrent [34]. BitTorrent supports direct download, if a resource is found it is copied to the requester. Only nodes that are interested get the resource.

Path Replication

In a multi-hop network where content is not transported directly such as in Freenet [102], it is possible to cache one replica of the content at each intermediate node. Since the intermediate nodes are acting as caches, path replication is also called cache-based replication. It is assumed that intermediate nodes provide storage space for replicas even if they are not interested in the content. Path replication leads to a high number of unused replicas.

Therefore, an improved approach replicates the content on an intermediate node according to a fixed replication rate (*path random replication* [101]). The advantage of this approach is a compromise between a higher replica usage and limited hop distance to other replicas. The difficulty of this approach is to specify a suitable replication rate for each file in advance, which is hard if the files are not known at

[3] This section is adapted from [7].

system startup.

An alternative is to specify a node specific replication probability, where nodes decide ad-hoc if a file is replicated or not. The replication probability is dependent on the peer's resource status and optionally refers to the replication rate, too. The authors in [101] refer this strategy to as *path adaptive replication*.

Active Replication

The goal is to place the right number of replicas at the right locations *before* they are requested. Researchers investigated the optimal number of replicas in the context of robustness. In [103] and [22] the authors investigate *random, proportional, and square root replication*. When applying random replication a uniform number of replicas for each object are created. Proportional replication creates replicas proportional to their query rate. The authors showed that square root replication determines the optimal replication rate r_i for object i, which is calculated as $r_i = \lambda\sqrt{q_i}$, with $\lambda = R/(\sum_i \sqrt{q_i})$, where q is the query rate and R is the number of object replicas in the system. Square-root replication does not consider the location of replicas. All strategies require global knowledge on the number of currently existing replicas and the current query rate for each of the replicas.

To reach square-root replication with limited knowledge researchers proposed *Pull-then-Push Replication* introduced in [104]. The first phase of this method regards the search of the content, with any existing algorithm. The second phase regards the replication of content to the neighbor nodes. To reach square root replication, the authors suggest that for the pull and push phase the same algorithms are used, because the number of replicas should be equal to the number of nodes visited during search. The authors evaluated typical algorithms, such as flooding and random walks. Their focus is set on robustness even on update situations. As multimedia content is usually not updated after creation and this algorithm only considers the number and not the location of content, this approach is out of scope of this work.

6.5.2. Proposed Replication Strategies

Although a unit is delivered hop-by-hop the basis for our proposed replication mechanisms is owner replication, since the units are consumed for some time at the requester and therefore need to be replicated to be further usable by other nodes. Units replicated at the requester can only be supportive for the immediate neighborhood. To

serve future requests, replicas should also be created on the delivery path. If the hormone concentration of a neighbor attracts a stored unit, the peer has to decide whether to move or to copy the unit. The simplest solution would be to apply path replication, but then the utilization of replicas would drop and the storage space is not used efficiently. Therefore, the goal is to find a replication mechanism that balances replica utilization and delay without the need of global information.

Beyond owner replication, path replication, and path adaptive replication we evaluate the following four replication mechanisms. These mechanisms exploit local knowledge on popularity and hormone values collected from neighbors.

Simple Hormone

If a unit is requested by peers from opposite parts of the network, the unit has to move first to one requester and afterwards to the other requester. This can lead to long traveling paths, which can be avoided by replicating a unit if more than one neighbor holds hormones for it. Note that it is not possible to differentiate if hormones on the neighbor are created by different peers. Thus, it is possible that unnecessary replications are made.

Local Popularity

Each node uses the local request history of the corresponding content to decide if it is likely to be requested again in the future. If the rank of the content is among the best 30 % the corresponding unit is replicated. So popular units are more likely to be replicated, but popularity information from neighbors is ignored. With this method the communication effort is minimized.

Neighbor Popularity Ranking

After collecting the popularity ranks for the content from the neighbors, the peer decides if it is worth to replicate the corresponding unit. The ranks are aggregated to a region rank (see Chapter 5), which is calculated as follows:

$$R = \frac{1}{n} \sum_{i=1}^{n} \ln(r_i)$$

n represents the number of neighbors and r is the rank of the specific unit at a neighbor i, where 0 is the best rank. To reduce the impact of peak ranks (e.g., one

unit is best ranked at two nodes, but worst ranked on the third node) the logarithm is used. If the region rank R is lower than a given threshold (e.g., the best 30 % at all neighbors) the unit is replicated.

Neighbor Hormone Ranking

Analogue to the popularity ranking the units can also be ranked by their hormone values at the neighbors. The higher the hormone value for a unit on a neighbor, the better is the unit's rank. The collected ranks can be aggregated as before and if the region rank is lower than a given threshold (e.g., the best 30 %), the unit is replicated.

6.5.3. Evaluation Settings

We extended our simulation by the replication strategies and added further evaluation support for peer-to-peer networks.

Network Topology

We assume for small overlay networks of 50 nodes a connected Erdős-Rényi random graph with a diameter of 6. For larger networks, e.g., with 1,000 nodes, we assume a scale-free network topology. The bandwidth was set to 100 Mbit/s; other bandwidth values are target of future work.

Initial Storage

Each node creates units until 30 % of each node's storage is filled, where each node provides 900 MB of storage size. At the beginning only one instance of each unit exists. We expect that in a scenario with 50 motivated persons, each person is contributing with equal probability. We generate 5,000 units for the scenario with 50 peers. We further assume that each person is represented by one peer.

The average size of a unit is 2.6 MB, whereas the maximum size is 16 MB and the minimum size is 190 KB, with a playback bit rate of 1 Mbit/s. These unit sizes are the result of the third SOMA use case "The long night of research" (see Chapter 2).

Request Generation

We kept the unit content model as described in Section 6.4 also the taste model of the clients did not change. However, a client can request a content array instead of a unit

η_0	η	α	ϵ	m	t	$maxhops$
3.95	4.39	0.45	0.16	0.23	0.23	10

Table 6.3.: Parameter settings

id, which allows keyword search. Additionally, in this scenario we do not consider any order of the requested units, thus, if a requested unit arrives, it is presented to the user immediately.

We further introduce a deadline for each unit, until which it has to be delivered. The deadline is dependent on the size, the link bandwidth and the maximum number of hops a unit can travel *maxhops*. If a deadline is missed, no further hormones for that unit are created to stop attracting content.

A request is considered as failed if none of the requested units could fulfill their deadline. A user can only submit one request at a time. If this request is fulfilled or failed, a new request will be generated.

Simulation Parameters

We used the genetic algorithm as described in Section 6.3. We change the fitness function to maximize the number of successful requests (i.e., consumed units within the deadline).

In Table 6.3 the resulting parameters are shown. The creation parameters η_0 and η are high, which leads to a wider travel range. The diffusion of 45 % of the hormones supports longer travel distances, too. The evaporation rate ϵ is in comparison to the creation value rather low, which means that the hormones last for some time. The migration threshold m describes the minimum hormone difference between two nodes to make a unit move. In this case the difference is very low in comparison to the values of η_0 and η. This leads to a very dynamic behavior of the units. The minimum hormone threshold t is the same as the evaporation value and causes hormones to stay long on the nodes.

Metrics

We want to evaluate the request fulfillment on the one hand and the utilization of replicas on the other hand. The fulfillment of requests is represented by the *delay*.

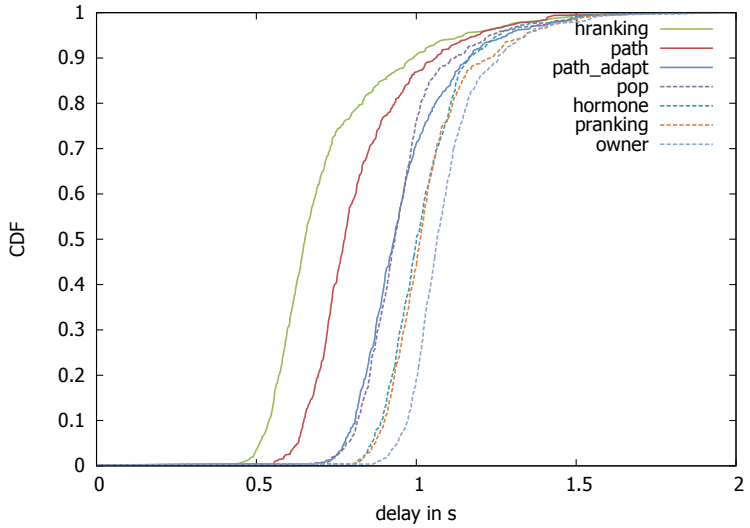


Figure 6.9.: Delay distribution in the best effort scenario [7]

The delay is measured from the request time of a unit until the arrival of that unit on the node. A delay of 0 s means that the unit was already on the node at request time. The delay is presented as cumulative distribution function (CDF) over the simulation time. The *deadline missed rate* represents the rate of units (not requests), for which the deadline is missed. If a unit missed its deadline, the delay is calculated as deadline minus request time (max. delay). The *request failed rate* indicates requests of which all units missed their deadline.

A unit is presented for some time, and we measure the rate of units that currently started with presentation. The more unit presentations started in comparison to the number of their replicas, the better the *unit utilization*. The utilization and the request failed rate will be depicted as box plot with 1.5 interquartile range whiskers.

6.5.4. Results

We conducted extensive simulations for the random and the scale-free network topology. We performed each simulation in 10 runs for 500 simulated seconds. Each run started with a different seed for the random number generator. The random number generator has an impact on the network topology and the request generation and anything further that needs random input. The results of these runs are averaged.

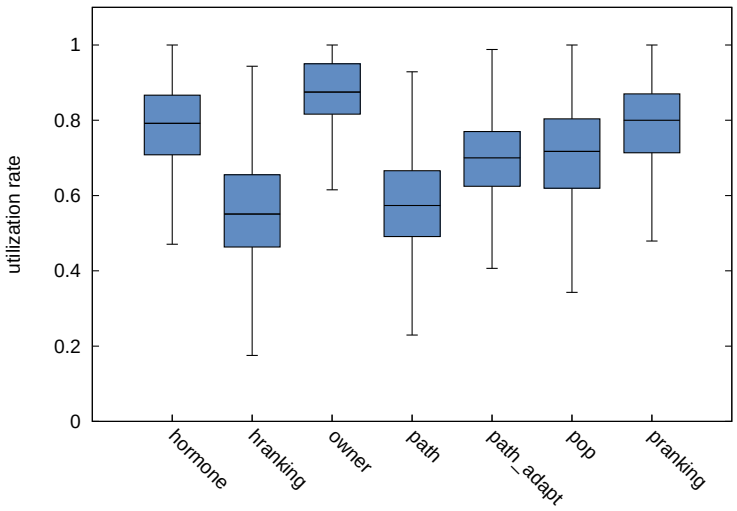


Figure 6.10.: Utilization comparison [7]

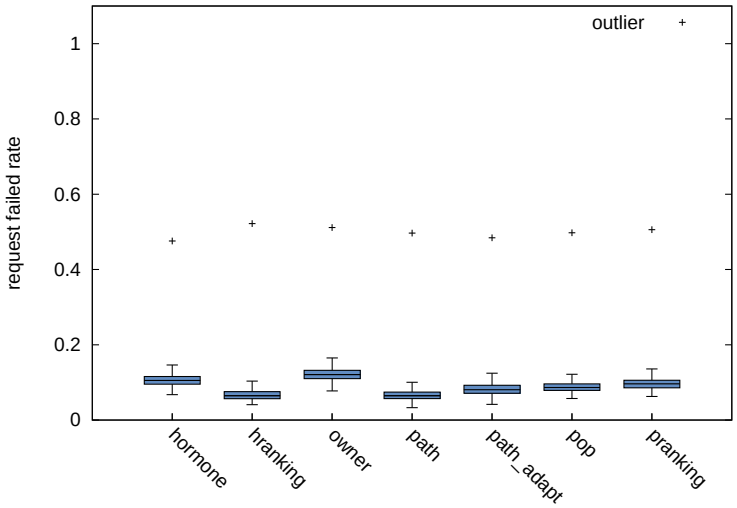


Figure 6.11.: Failed request rate [7]

The delay stabilizes after an initial simulation time of app. 100-200 seconds, a run time of 500 seconds for the random network is therefore sufficient.

In this part we evaluate the impact of replication, which takes place in a best effort manner until no storage space is available anymore. A full storage space on a peer prevents forwarding of further units. The goal is therefore to find an intelligent replication mechanism.

In Figure 6.9 the delay development for all replica mechanisms is depicted. Neighbor hormone ranking (hranking) outperforms path replication (path), although path replication generates more replicas. The placement of hormone ranking is more efficient, leading to less overloaded nodes and therefore more requests can be fulfilled. Local popularity (pop) has a lower delay than popularity ranking (pranking), because the best 30 % for popularity may contain more units than the best 30% of popularity ranking, thus popularity ranking creates fewer replicas. An adaptive threshold for popularity ranking might lead to better results. The path adaptive replication mechanism (path_adapt) shows that random decisions can also lead to good results. The hormone replication mechanism (hormone) tries to replicate if there are currently further requests for that unit from somewhere else. This leads to a low number of replicas if the hormones do not reach the current location of the unit. This graph shows that the number of replicas has a high impact on the service quality. If too many replicas are created, the storage used inefficiently and if full, it blocks further transport. If the number of replicas is too low, the delay is high, because of long distance transport.

A more detailed view on the storage efficiency is shown by the utilization rate. Note that it is collected only once, at the playback start. Figure 6.10 shows interesting results. Owner replication has the best utilization, which is explainable by the low number of replicas. On the first sight one might say that the more replicas created the lower the utilization. But the hormone ranking mechanism creates fewer replicas than the path replication mechanism. The difference in utilization can be explained by comparing the delay of these replication mechanisms. The hormone ranking replication has a far lower delay than the path replication. Since a request is sent after another is fulfilled or failed a lower delay means that more requests can be submitted and therefore more replicas can be generated, which results in lower utilization. In general the utilization is a metric that has to be evaluated in combination with delay and request failed rate.

The request failed rate is depicted as box plot in Figure 6.11. It shows that hormone ranking and path replication perform 50% better in comparison to owner replication. This indicates that the placement of units is well developed. The outliers (marked with x in the figure) are experienced during the first few seconds of the simulation, when the unit placement is random. After that the failed requests go fast down to approximately 5-10 %.

6.5.5. Discussion

Uninformed replication such as path replication is resource-wasteful. Path adaptive replication showed to be a good compromise. Thus, for networks without knowledge about current demands, path adaptive replication can be recommended. The evaluation of local popularity replication and popularity ranking showed that popularity aging has to be considered since in dynamic environments the popularity can change quickly. Therefore, we address the hormone as dynamic and latest information about popularity of the content. By additionally including the neighborhood into the decision, the performance increases as well.

An ideal replication mechanism would be lowest in delay and best at utilization. Until now, none of the described replication mechanisms matches this pattern. Thus, there are further strategies necessary. As an example, clean-up mechanisms can be used (such as least recently used) to increase the utilization, by taking care of unneeded units. Additionally, storage load balancing would increase the reliability of the system.

6.6. Storage Balancing by Introducing Clean-up Mechanisms

We concluded before that efficient replacement or clean-up has to be done in order to avoid blocking the transport of units and to increase the utilization of replicas. A clean-up is triggered if a certain storage level is reached, which leads to a balanced storage load of the system[4].

[4]This section is adapted from [8].

However, if we want to guarantee that always at least one instance of the unit is available in the system, the decision of what unit to delete becomes more complicated. Therefore, the goal is to find an efficient strategy that does not influence the delay, but on the other side increases the utilization of replicas. We apply the basic principle that a unit can be deleted only if there is a copy of it at one of the neighbors.

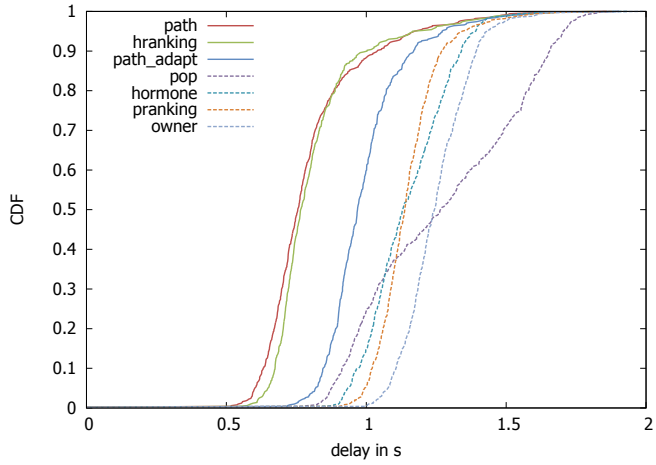
We compare three mechanisms: least recently used (LRU), least frequently used (LFU) and hormone-based clean-up. We do not evaluate smart clean-up (see Section 6.4) anymore, since the high number of movements has a bad impact on the delay. LRU takes care of popularity aging. If a unit is not popular anymore its replicas can be removed. LFU targets the cumulated popularity, which causes less effort than LRU. We introduce hormone clean-up, which exploits local knowledge about hormone concentration. A unit is deleted if there are no hormones for it on the neighbors, thus there is currently no demand for it. So, units currently in delivery are not deleted.

We apply the clean-up to the before analyzed replication mechanisms in the following two scenarios: (1) A 50 node random network and (2) a 1,000 node scale-free network. We further analyze the impact of peer churn to the replication and clean-up mechanisms.

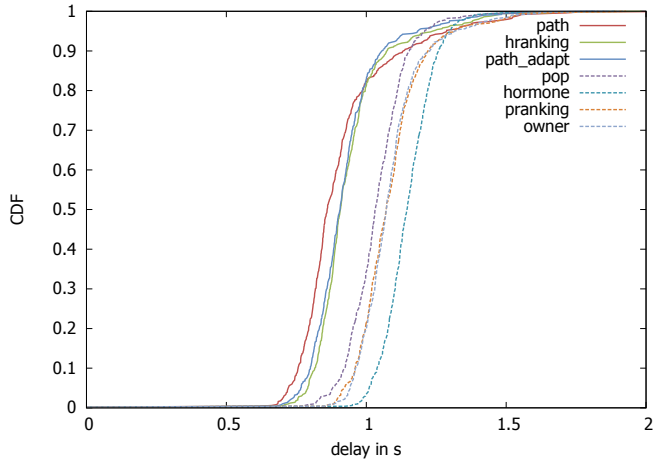
The settings are the same as in the replication scenario, but we add the clean-up threshold c=60%. If the storage usage of the node reaches c, the configured clean-up method is triggered. To generate a scale-free network the Eppstein Power Law Algorithm [105] is used. The algorithm gets as input a random graph and by repetitively removing and adding edges a power law distribution is reached. The network diameter of the scale-free graph is 13 and we extend the simulation time to 700s. The scale-free scenario starts with 15,000 units in comparison to 5,000 units for the small network.

6.6.1. 50 Nodes Random Network

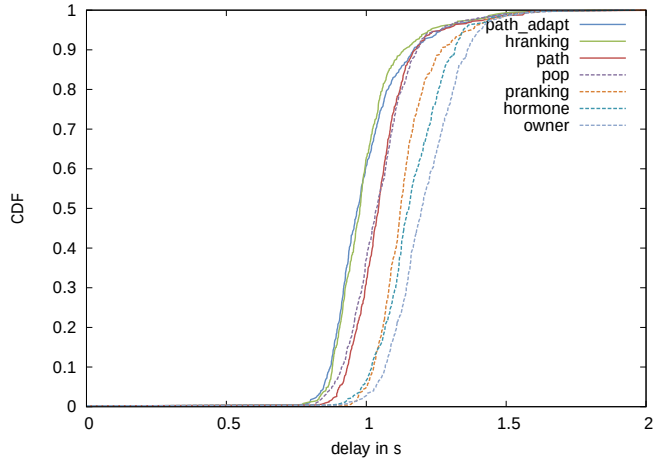
In comparison to the pure replication shown in Figure 6.9 in Figures 6.12(a), 6.12(b) and 6.12(c) hormone clean-up, LRU and LFU are applied. One can see immediately that some combinations of clean-up and replication are advantageous, whereas some combinations lead to performance drops. The most conspicuous example is the combination of local popularity replication (pop) and hormone clean-up as shown in



(a) Delay distribution of hormone clean-up



(b) Delay distribution of LRU clean-up



(c) Delay distribution of LFU clean-up

Figure 6.12.: Delay comparison of the different clean-up mechanisms [8]

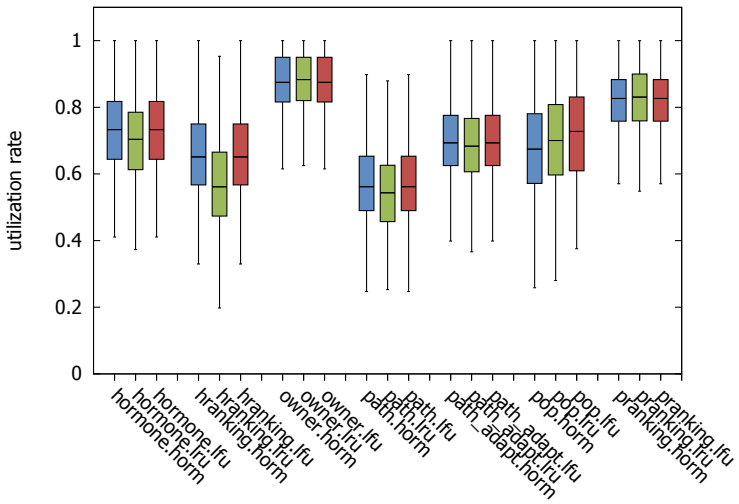


Figure 6.13.: Hormone, LRU and LFU utilization comparison [8]

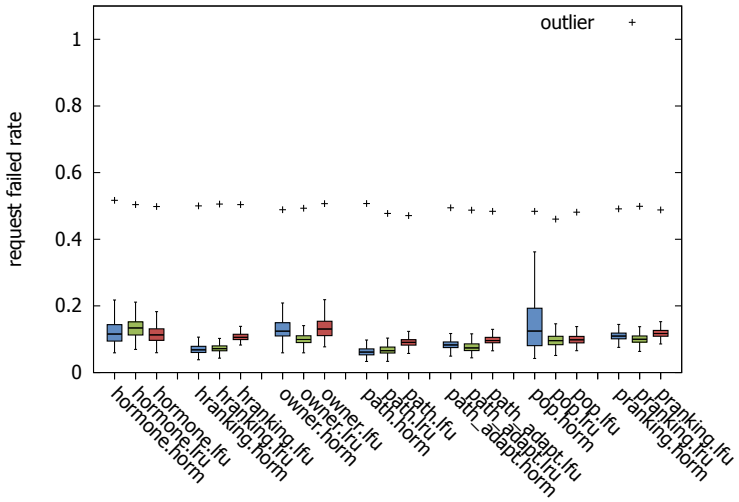


Figure 6.14.: Hormone, LRU and LFU request failed rate comparison [8]

Figure 6.12(a). The reason for this bad performance is that the hormone clean-up only considers current demands for a unit and therefore does not fit the replication mechanism.

It is further interesting to see that path replication in combination with hormone clean-up results in a lower delay than hormone ranking. Hormone ranking initially places units more efficient, leading to fewer replicas than path replication would create. The downside of hormone ranking is that popular nodes evolve, which are filled first. This leads to blocked transport paths, which in turn increases the delay.

The LRU clean-up depicted in Figure 6.12(b) has the most positive impact on path-adaptive replication since the delay is stable in comparison to the best effort scenario, while the other replication mechanisms experience higher delay. Path replication has the lowest delay, but also results in a flatter curve than path-adaptive replication and hormone ranking, indicating a higher delay jitter.

LFU (Figure 6.12(c)) leads to a high number of wrong clean-up decisions, because the delay increases for all replication mechanisms. Especially the hormone replication mechanism and owner replication suffer from wrong decisions; the delay jitter increases. Path replication results in doubled delay. The local popularity mechanism experiences positive impact, since LFU also prioritizes popular units.

The utilization measures the improvement in storage efficiency of the single replication mechanisms (see Figure 6.13). All clean-up strategies lead to an increase of utilization. Hormone replication does not take advantage of a clean-up, because it already creates only a low number of replicas. Hormone ranking has the worst utilization if combined with LRU, but has even in that case a higher utilization than path replication. Local popularity replication results in a high variance, which also shows that some improvement is needed.

A high utilization does not necessarily indicate that the replication mechanism is best suited for the delivery. A high utilization can also be reached if deadlines are missed and therefore the delivery did not take place, which leads to a lower number of replicas. Therefore, it is important to aggregate the information of delay, utilization and the requests failed statistics.

In Figure 6.14 one can see immediately the low performance of local popularity

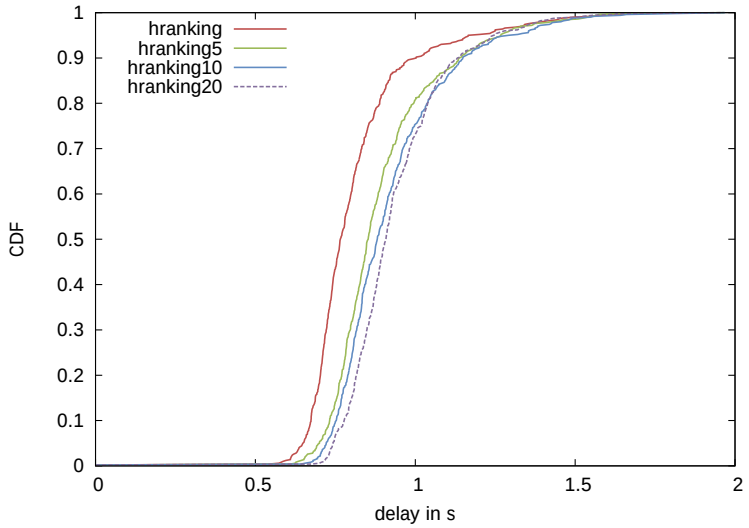


Figure 6.15.: Delay distribution of hormone ranking with hormone clean-up if 5,10, 20 nodes fail [8]

replication if combined with hormone clean-up. The lowest variance is reached by combining it with LFU. The other replication strategies do not profit from the combination with LFU, the request failed rate is the highest. Most replication strategies work best if combined with LRU, except hormone ranking.

The decision of what clean-up strategy to choose should be made by combining the results of delay, utilization and request failed rate. For example, path-adaptive replication should be used with LRU, because of its steep delay curve and low failed rates. Hormone ranking replication should be combined with hormone clean-up. Path replication should not be combined with LFU, because of low delays. Path replication with hormone clean-up leads to a higher utilization, but also to a higher failed request rate than if combined with LRU. Furthermore, it is important to evaluate the robustness of the before mentioned replication and clean-up combinations.

6.6.2. Impact of Peer Churn

We simulate churn as randomly chosen nodes being removed periodically one-by-one. We do not handle isolated nodes after peer deletion. Thus, there is a performance gain potential if an overlay algorithm takes care of reconnecting such nodes. We chose

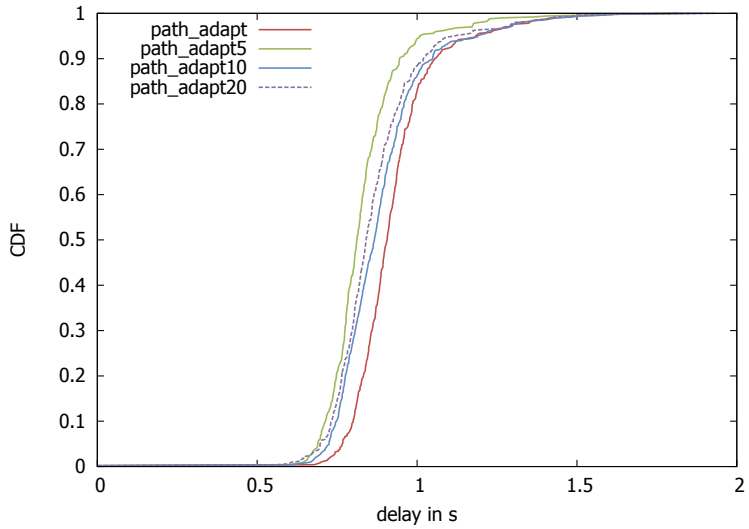


Figure 6.16.: Delay distribution of path adaptive with LRU if 5, 10, 20 nodes fail [8]

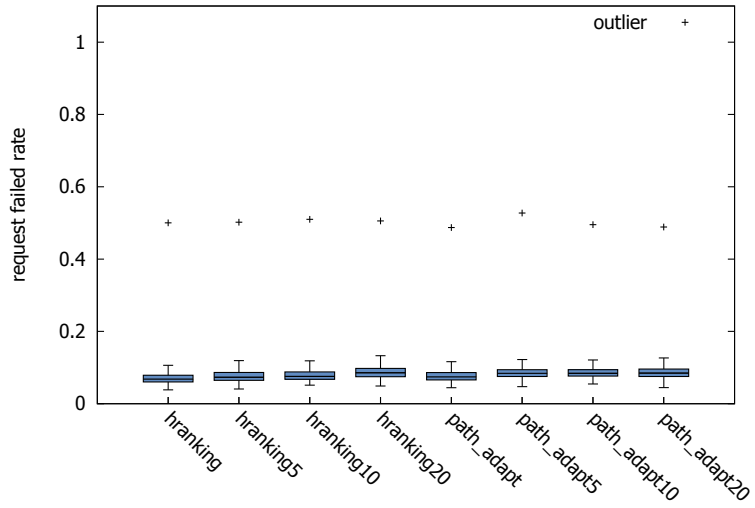


Figure 6.17.: Failed request rate in case of peer churn [8]

5, 10 and 20 nodes to be removed. We compare hormone ranking with hormone clean-up and path adaptive replication with LRU.

Figure 6.15 shows the delay distribution of the hormone ranking algorithm. One can see that the replication algorithm and the delivery algorithm are capable of handling loss. Overall the delay increased a bit, but interestingly the delay for 5, 10 and 20 removed nodes is very similar. However, the failed requests increase slightly in both hormone ranking and path adaptive scenarios, as shown in Figure 6.17. In general if a keyword is matched by a number of units, a system wide loss of a unit does not have a major impact.

The path adaptive algorithm leads to interesting results as shown in Figure 6.16. Here, the delay of the peer churn scenarios is lower than in the original case. This anomaly can be explained by referencing the clean-up failures of the original scenario. A clean-up fails if on the current node all units are currently in use or there is no copy of the current unit on a neighbor. A disadvantageous replica distribution might be that at every second hop a replica is placed, which means that a high number of replicas exist in the system, but because the nodes only see their neighbors, the units cannot be deleted. If a peer fails, a unit has to move an alternative way. Therefore, the unit movement is increased and more requests can be fulfilled. Path adaptive replication might need an alternative clean-up policy taking a larger neighborhood into account.

6.6.3. 1,000 Nodes Scale-free Network

In this section we evaluate the applicability of our delivery algorithm for scale-free networks. We evaluate hormone ranking with hormone clean-up and path adaptive replication with LRU. It is shown that the parameters for the 50 node network also work for the 1,000 peer network. Specific optimization using the genetic algorithm could lead to even better results.

In Figure 6.18 it is depicted that the delay is increased by around 500 ms in comparison to the small network for the hormone ranking algorithm. Furthermore, if 100, 200 and even 500 nodes fail the delay does not increase considerably. Note that also high degree nodes may fail, because the nodes leaving the network are chosen randomly. Figure 6.19 shows that the problem with clean-up failures is not experienced as in the 50 nodes network, which can be explained by the network structure and its rather low

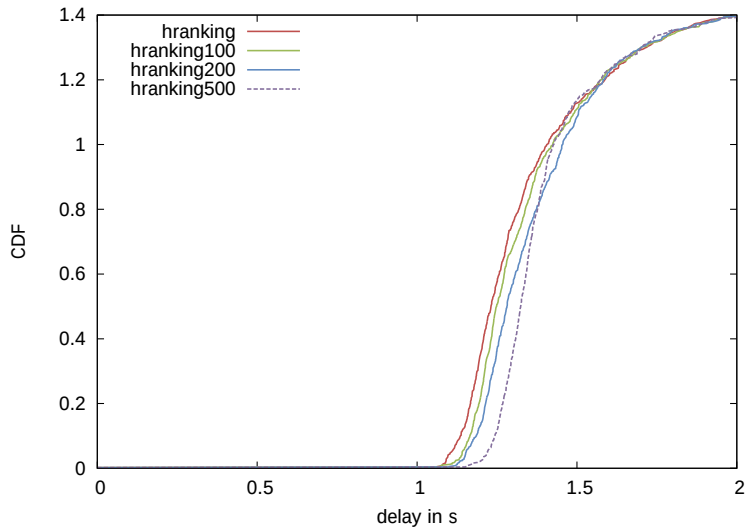


Figure 6.18.: Delay distribution of hormone ranking with hormone clean-up if 100, 200, 500 nodes fail [8]

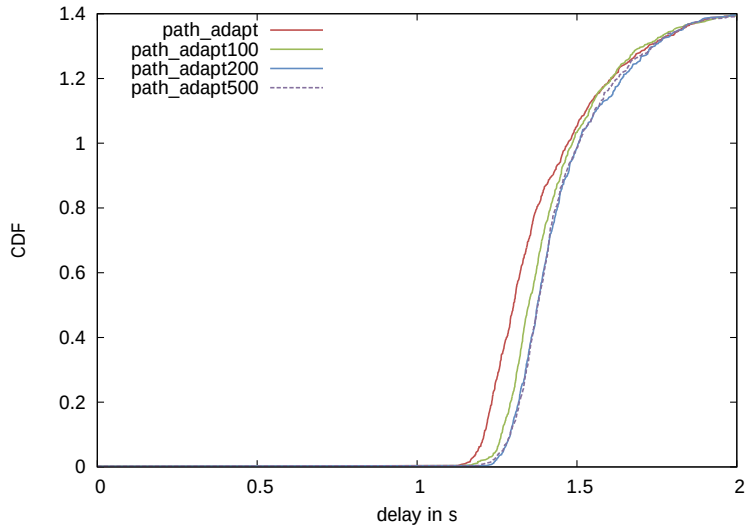


Figure 6.19.: Delay distribution of path adaptive with LRU if 100, 200, 500 nodes fail [8]

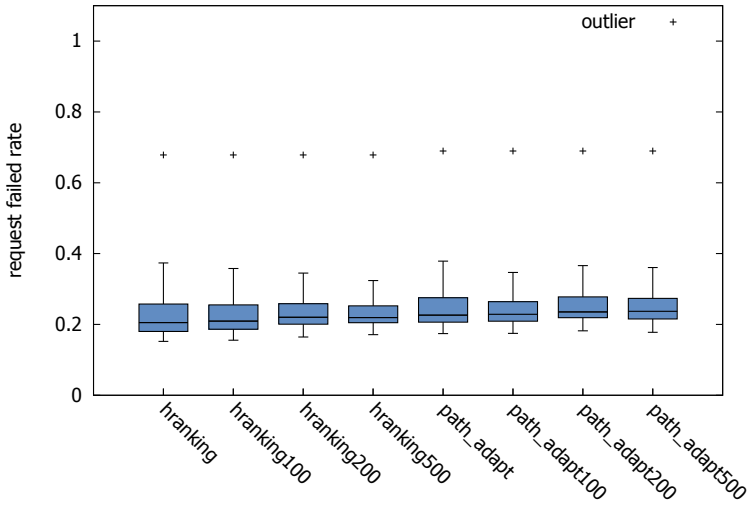


Figure 6.20.: Failed request rate in case of peer churn [8]

diameter. Therefore, the delay of path adaptive replication is similar to the hormone ranking algorithm. Both algorithms show slight increases of request failures also in the presence of peer churn (see Figure 6.20).

6.6.4. Discussion

Although the results of the replication mechanisms are promising, the clean-up has a negative impact on the delay, which means that the goals of the clean-up introduction are not reached. Ideally, a combination of replication and clean-up should lead to low delay, low failed rate and high utilization. If the settings are as strict as in this section, path replication with hormone clean-up, although inefficient, performs best. Alternatives could be path adaptive replication with LRU and hormone ranking replication with hormone clean-up. The node failure scenarios showed that there are still nodes, which block the transport of units. To solve this issue the settings could be less strict regarding the deletion policy. Instead of deleting a unit only if there is a copy of it in the neighborhood, it could be weakened to delete a unit if another unit covering the same keyword is in the neighborhood.

6.7. Summary

This chapter described the development of an artificial hormone system that allows for efficient delivery of multimedia content. The algorithm allows for handling different request types, i.e., sequential, parallel and mixed requests. It is further QoS-aware by regarding link qualities. The algorithm can be considered as adaptive, robust and scalable, which has been shown in different scenarios. We further investigated different replication mechanisms that create replicas on the travel path. This measure does not only increase the robustness of the system, but also reduces the experienced delay for the clients. In an ideal situation the content is placed on the nodes before it is requested. Since we expect limited storage space, we further extended the algorithms by clean-up mechanisms that on the first hand should not increase the delay and on the other hand take care of storage balancing.

7. Artificial Hormone Systems as a Middleware for Content Delivery

In the previous chapter units comprised videos with audio track or photos and metadata — multimedia units. However, more and more systems transfer their storage of mixed content (web sites, documents, services, etc.) to the cloud. In the cloud the resources are typically distributed in a network consisting of a high number of nodes. The nodes and connections form a complex network, where the control of resource placement and relocation is not feasible with a centralized algorithm. Beyond this, we want to generalize the application of our algorithm and therefore we propose a middleware that covers the transparent delivery and placement of content[1]. The basis of the middleware forms the hormone-based algorithm introduced in Section 6.2.

We show the applicability of the middleware by conducting a case study. The case study targets multimedia sharing, however, is also used to make recommendations on unit sizes for other applications.

7.1. Related Work

Brinkschulte et al. adapted their artificial hormone system for task allocation [94] to be used as a middleware for organic networks [106] and applied this middleware to a real scenario for automated guided vehicles (AGV). The AGV carries a number of sensors which have to be coordinated to guide the AGV through the environment and help performing its assigned tasks.

Balasubramaniam et al. derived a communication resource management system inspired by the blood glucose regulatory model of the human body [107]. Their approach aims at IP networks and provides resource allocation strategies at short-term, mid-term and long-term levels to support Internet Service Providers (ISP) and

[1] This chapter is adapted from [99]

the underlying carrier operators. The results indicate that a bio-inspired resource management system can overcome traditional resource management strategies such as static and weighted fair sharing approaches.

Tempesti et al. propose an architecture supporting the formation of large complex networks [108]. The architecture defines simple computing units (ECells) that are connected by a high-speed serial communication protocol (EStack). This work is orthogonal to the work presented by Balasubramaniam, which is based on standard communication networks.

Harsh, Chow and Newman argue that the network structures of the human brain can be a reference model for bio-inspired networks. Their gray networking model [109] refers to networks that implement a significant subset of features, among them multilevel and multiple feedback loops, multiple sensors and a filtered inter-component communication. By this definition, our proposed approach can also be viewed as a kind of gray networking.

7.2. Artificial Hormone System Middleware - MASH

The proposed MASH (Middleware using Artificial Systems of Hormones) is responsible for providing delivery and communication in a dynamic network without central control, and is adaptive, robust and scalable. It manages seamless interaction between content of the nodes and guards the content through the network. The application only has to provide the request in the correct form and the network has to provide information about the direct neighbors of each node. The MASH can be applied on mobile and wired devices. Figure 7.1 depicts an overview of a sample network with the installed middleware. Each node that is part of the network can request content and provide content. The idea is to self-organize the placement of the content in such a way, that the traveling distance of content is reduced, ideally to zero hops. For that reason, it is necessary that a node can express its demands, that content can be distributed freely (although with space restrictions) and that the storage is managed efficiently. These tasks are brought together by multi-hop communication, thus, a node only needs to know its neighbors. The artificial hormone-algorithm is used to fulfill the before mentioned requirements.
As an interface language between application and delivery mechanism we propose to use ViNo (see Section 4.2), even if the timely behavior is of secondary importance.

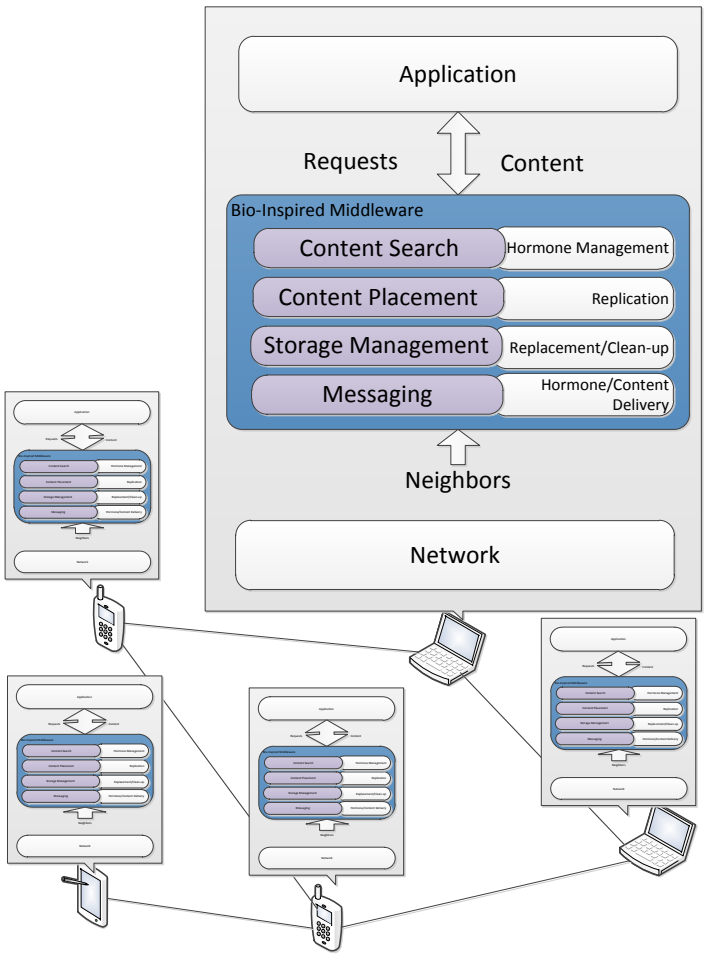


Figure 7.1.: MASH Structure

The application can use the wildcards including metadata description for search. The delivery returns the matching units. For sake of simplicity we define that a unit substitutes a wildcard if all tags of the unit are equal to the required tags. If the application requires other wildcard matching principles, this has to be configured before system start up. For a real implementation the MASH can provide different matching profiles that have to be selected by the application. The specific interfaces to the application and the network are defined in Appendix B.

7.3. Case Study

In this section we apply the middleware to a real scenario. Our scenario is a multimedia sharing system to be used at social events.

With the MASH the visitors are able to share their most interesting multimedia content with other visitors during the event. The application provides a graphical user interface that allows for the definition of requests and also to provide own content (composition and decomposition) [5]. A request is communicated to the MASH by using ViNo. We assume an underlying ad-hoc wireless network, since users are moving around the area of the event.

The case study has two goals. First, it shows the applicability of the MASH to a real scenario. Second, we further use the scenario to make parameter recommendations for other use cases than continuous data delivery. In particular, we evaluate typical unit sizes from 56KB up to 512KB per request. One might say that the same results apply for unit sizes as for chunk sizes of BitTorrent[110]. However, BitTorrent assumes full download of large content of several hundred megabytes, whereas we assume first that the consumed content is smaller and that a unit is played immediately after reception (progressive download). Since consumers of videos are very sensible regarding delay, this is a very good scenario to make recommendations for more general purposes if using MASH.

7.3.1. Settings

We apply the content model as already described in Section 6.5. A client selects tags out of a set of predefined tags and creates a request. The request is sequential and matches a playback duration of 30s with a playback bitrate of 200 Kbit/s. So, a unit size of 512KB results in 2 units per request. Additionally, a client adjust its

	η_0	η	α	ϵ	m	c	t
512KB	1.31	3.26	0.49	0.01	0.01	10%	0.08
256KB	1.23	2.99	0.45	0.03	0.01	10%	0.03
128KB	1.36	3.46	0.49	0.03	0.45	10%	0.58
56KB	1.98	0.32	0.41	0.53	0.59	10%	0.28

Table 7.1.: Parameter settings

taste to a currently watched unit with a probability of 10%. We further introduce a deadline for each unit to be delivered, and we set it to 40ms, which is the desirable delay between two frames. If a deadline is missed, no further hormones for that unit are created to stop attracting content.

At system startup a network of 100 nodes is generated with a link speed of 1 Mbit/s. We assume that the provided storage space of each node is strictly limited to 900MB. So we set up the system with approximately 1% of a node's provided storage is filled with units (which follow a Zipf-like distribution regarding their tags). This ensures the same storage size for all unit size scenarios. The number of units is unit size dependent, where the following numbers of units are generated: 16100 for 56KB, 7100 for 128KB, 3600 for 256KB, 1800 for 512KB. In a scale-free network the 10 nodes with the highest number of connections create more units, which results in 17710 for 56KB, 7800 for 128KB, 3950 for 256KB, 1980 for 512KB. We use hormone-ranking replication in combination with hormone clean-up and replacement. A unit can be deleted if another unit with the same content is placed on a neighboring node.

We use the genetic algorithm as described in Section 6.3 and list the actual parameters in Table 7.1. As input for the genetic algorithm a connected Erdős-Rényi random graph of 100 nodes was chosen and a unit size of 512KB, the request playback length is reduced to 10s (i.e. 2 units per request). The resulting parameter set obviously prioritizes the 512KB scenario, but showed good results for all unit sizes. We therefore took this parameter set as starting point for the other unit sizes and started the genetic algorithm again. All parameter sets seem to be similar, except the 56KB parameter set, because in this scenario the clean-up threshold is never reached.

7.3.2. Results

We evaluate the different file sizes regarding delay. We concentrate on the inter-request delays and omit the delay of the first unit of a request (startup-delay), which

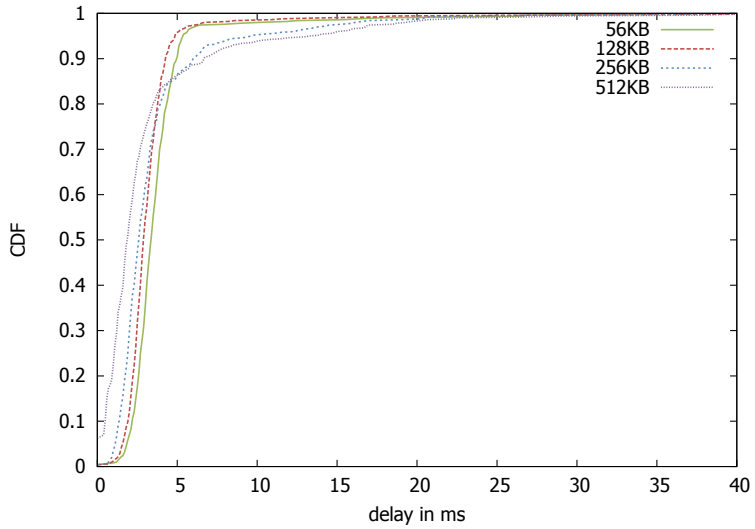


Figure 7.2.: Inter-request delay distribution

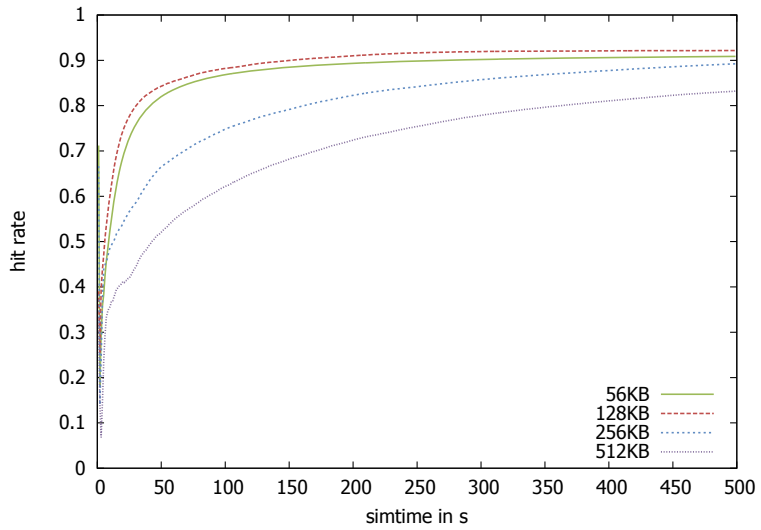


Figure 7.3.: Hit rate of different unit sizes developed over simulated time

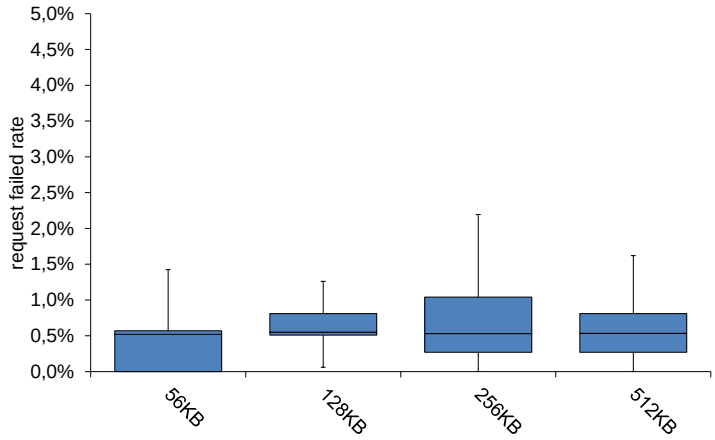


Figure 7.4.: Boxplot with 1.5 inter quantile whiskers of the deadline misses

is obviously unit size dependent. Furthermore, we analyze the rate of failed requests and the hit rate.

First, we analyze the random network scenario and then compare it to the scale-free scenario. In Figure 7.2 the inter-request delay is presented. It can be seen that the 512KB scenario shows the best delay, but the delay flattens, which indicates a high delay jitter. The highest delay is experienced in the beginning of the simulation, where the units are not placed optimally yet. The delay stabilizes within the first 100 to 200 seconds. For larger units, the placement takes longer than for small units, and therefore the delay jitter is larger for 512KB and 256KB. The 128KB scenario seems to be exactly between the 256KB and the 56KB scenarios. It shows the low delay of the larger unit sizes and the low delay jitter of the small unit sizes. It generates faster more replicas than in the 56KB scenario and results therefore in a lower delay. Additionally, the migration threshold m of the 128KB scenario lets units remain longer on nodes.

In Figure 7.3 the hit rate development during the simulation time is depicted. The placement of the units evolves to a very high hit rate up to 90%, thus, in 90% of the cases the units are already at the node, where they are requested. The 128KB units even have a slightly higher hit rate than the 56KB units, because at startup

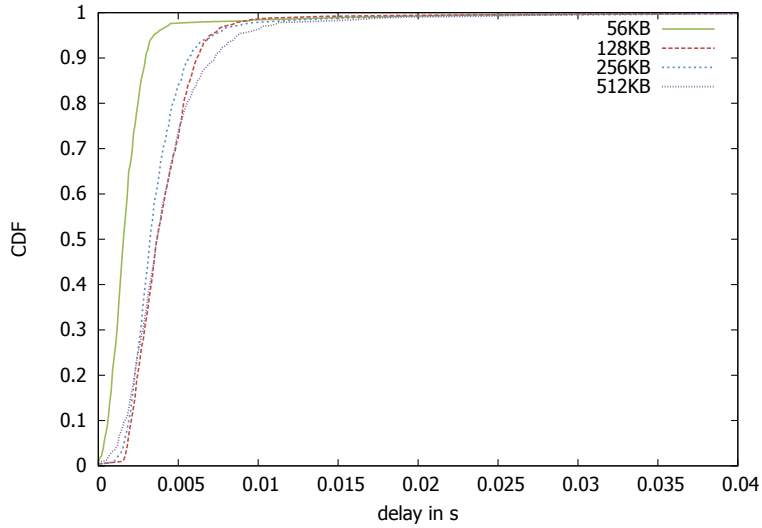


Figure 7.5.: Inter-request delay distribution

more replicas of the 128KB units are created. By the time the clean-up mechanism balances the number of replicas and therefore also the hit rate remains stable. The difference of the 512KB units to the other unit sizes is during the first hundred seconds more than 20%, which shows how long it actually takes to place the units.

Finally, we compare the scenarios regarding their failed request rate. A request is failed if the deadline of each unit of the request is missed. In Figure 7.3.2 one can see the results. All scenarios show the same median of failed requests, which is about 0.5%. However, the differences can be seen in the variance of the results. Here, the 56KB scenario shows the best results, since only a few requests fail in the beginning. The 256KB scenario has a large variance of failed requests, although having a low delay. The reason is the clean-up mechanism, which deletes too many replicas at a certain point of the simulation. This increases for a short time the deadline misses and therefore also the variance of the request failed rate is changed. An adaptive threshold for the clean-up should be considered.

In the following we evaluate the scale-free scenario with the same parameter sets as before. In such a network some of the nodes have more workload to handle than other nodes, but on the other hand provide twice storage space of normal nodes. We call these nodes hubs. In such a system the unit size can have a large impact on its

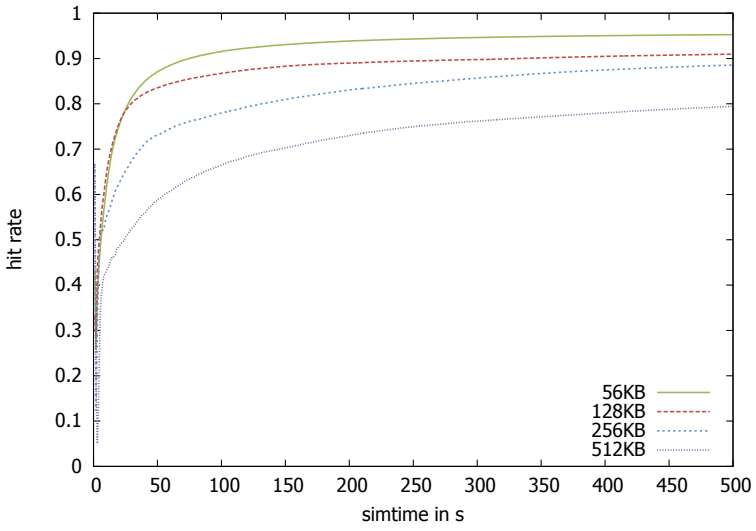


Figure 7.6.: Hit rate of different unit sizes in a scale-free network

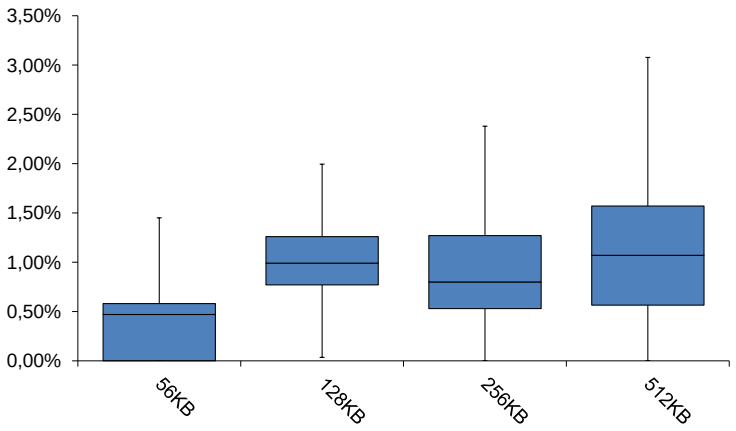


Figure 7.7.: Boxplot with 1.5 inter quartile whiskers of the deadline misses in a scale-free network

performance. In Figure 7.5 it is shown that the inter request delay differs from the random network scenario. The performance increases if the additional storage space on the hub nodes can be exploited to create more replicas and if these replicas are not deleted by the clean-up mechanism. The 56KB scenario can take this characteristic as an advantage and therefore its delay is the lowest. The other unit sizes reach a similar delay, but the 256KB unit size does not lead to a high delay jitter such as in the random network case. That the 56KB unit size performs best is also shown in the hit rate comparison in Figure 7.6. The 56KB units lead to an hit rate increase up to 95%, whereas the hit rate of the 512KB unit size decreases. The request failed rate doubles for all unit sizes except for the 56KB units. The variance of the 512KB units shows that there are request misses up to 3%. The 256KB units lead regarding the median to a lower failed rate than the 128KB unit sizes, however, the variance of the 256KB units is higher. The reason is that in the 128KB scenario more replicas can be created on the hub nodes.

7.4. Summary and Discussion

In this chapter we introduced MASH, a middleware for content delivery, which targets dynamic networks by providing robustness, adaptability and scalability. As a basis we used our proposed hormone-based delivery algorithm and defined interfaces for connecting the delivery to users and to the network. We conducted a case study that shows the applicability of the middleware in a real scenario. We applied the study to random and scale-free network topologies. The parameter sets collected in this case study make MASH widely applicable. However, one of the most critical parameters for content delivery, the chunk size, has to be application dependent. For making recommendations on this parameter we used the performed case study to evaluate typical unit sizes from 56KB to 512KB. It is noteworthy that MASH works with all sizes and the placement of units stabilizes by the time. However, performance differences might have an impact on the quality of the application. For scale-free network topologies chunk sizes of 56KB showed the best results. The typical structure with high degree nodes supports the small chunk sizes. In a random network 128KB chunk sizes perform better, because of longer travel paths and time spent on the nodes. In multimedia applications it is further advisable to use smaller units, because the user experiences larger gaps if large units are lost. In general if the content to deliver is small, it is useful to use small chunk sizes. Another point is the size of the metadata. In our case study metadata size is negligible and

therefore the management effort limited. If content with large metadata information has to be delivered, the performance might be different and larger unit sizes are recommendable.

8. Conclusions and Future Work

In the context of the Self-organizing Multimedia Architecture Project (SOMA), this thesis covers delivery solutions for non-sequential media access in dynamic networks. *Non-sequential media* access can be seen as a composition of different multimedia parts (video units) that can be consumed in parallel, sequential or mixed parallel and sequential. For the composition definition and for the description of the delivery of such compositions a formalism has been introduced, which is referred to as *VideoNotation (ViNo)*. It has been shown that this description language can be applied to many use cases, such as the definition of video presentation, the definition of requests and the description of video delivery. ViNo is capable of describing provided and required QoS and based on that to perform simple calculations for supporting system designers in their decisions. This has been shown by comparing simulations of content delivery network with ViNo calculations.

The delivery of non-sequential media brings new challenges. Typical caching policies for videos are not valid anymore, because non-sequential media does not follow the traditional sequential consumption pattern. It is not defined what is the beginning and the end of non-sequential media. Therefore, new caching techniques were introduced and evaluated. By assuming that some combinations of units are more popular than others, the caching technique calculates a popularity rank of the current unit and if this is popular enough it is cached. The hit rate and delay evaluations showed the applicability of the popularity rank caching.

However, popularity is a global measure. If non-sequential media has to be delivered in dynamic networks, the delivery has to be adaptive, robust and scalable and without global control. Self-organizing algorithms have these attributes per definition and therefore the main part of this thesis concentrates on *self-organization*. We applied a typical design technique for self-organizing algorithms by abstracting phenomena found in nature. The proposed approach is inspired by the human endocrine system. It combines search and transport with the help of hormones. Hormones are created and spread over the network indicating the demand for specific content. The content

routes through the network towards a higher hormone concentration, where the highest concentration is found at the requester. Negative feedback is implemented by the evaporation of hormones and by deletion of hormones if content is available on the current node. As a basis for the algorithm an unstructured overlay is assumed. The nodes only know their neighbors and therefore all decisions are performed locally. The local decisions force the placement of content according to the needs of the client. This is shown by delay stabilization after a few seconds, if the system starts with random placement. The algorithm is further QoS aware, since a node can decide how many hormones should be spread to which neighbor, thus guiding the content over the best path. If an intermediate node fails, the content may be routed over alternative paths.

If a peer-to-peer overlay is assumed, the search space can get large and may have a bad impact on the delay. Additionally, content may disappear if nodes fail. Therefore, we investigated different replication techniques to leave replicas on the distribution path. The replicas reduce the delay for future requests of intermediate path nodes. One goal of the replication strategy was to efficiently use the provided storage space. We compared existing replication techniques with new techniques based on local popularity and hormone knowledge. We adapted the caching policy introduced before to aggregate the hormone rank within the neighborhood. If the demand for the content is high enough it will be replicated. However, the replica utilization in combination to the delay was low, therefore clean-up mechanisms were applied to increase the efficiency of the replicas.

It has been shown that the algorithm helps to deliver and to place non-sequential media efficiently and QoS-aware in a dynamic system. In combination with replication and clean-up units place themselves in the right places. The algorithm considers limited storage space, is adaptive in case of peer churns or other influences in the network, works in random and scale-free network topologies and copes with different user and request models.

In [3] we identified a number of requirements for a SOMA delivery sub-layer.

- A compliant delivery should be aware of non-sequential video access. It should use this knowledge in order to make the delivery process efficient.
- Since content is produced and consumed at different places by different clients

in a very dynamic way, it is common that nodes fail. The content, however, still has to be accessible. Therefore, the delivery model has to be robust.

- Although the content is divided into small units, temporal constraints have still to be fulfilled. Thus, the content has to be delivered according to a given QoS

By the application of our hormone-based algorithm all of these requirements are met. We further integrated the delivery algorithm into a middleware (MASH) that provides an interface to the application and the network and thus can be seen as the distribution layer of SOMA. A case study on different unit sizes allows for recommendations for the delivery of content different from videos and photos.

In conclusion this thesis showed that the characteristics of self-organizing algorithms are advantageous for non-sequential media delivery. The simple rules implemented by each node help to cope with the complexity of a dynamic scenario. However, in systems where there is no need for adaptation to a new situation, self-organizing mechanisms can be inefficient. Beyond the advantages of self-organizing systems of being robust, adaptive, and independent of global control, however, this could be also disadvantageous [10]. Self-organizing systems are hard to control and often their actions are hard to predict. A simple change from outside can cause either no effects or large effects. Another problem is trust. Because of the unpredictability one would never accept a self-organizing safety system, e.g., in a car.

From these considerations a number of research questions are matter of future work:

- To allow free usage of ViNo, we left the definition of its semantics open. However, to improve the usability of ViNo different patterns should be investigated that can be used out of the box.
- We used different client models and modeled their taste and request generation based on our assumptions. We want to further evaluate the interaction behavior of the SOMA use cases to get new client models and evaluate their effects on the delivery.
- The *Long Night of Research* offered the possibility of collecting consumption data. Visitors provided videos and photos of booths and talks they were interested in. We want to build a map of *interestingness* to conclude for future events how different topics should be arranged to attract the most visitors.

Additionally, we can find out where hot spots emerge and investigate their effects on the delivery.

- Parts of the algorithm are implemented for a real setting - see SOMA use case World Games. However, a realization of a mobile application that uses the middleware is still open. Although the visitors at our use cases were very interested in providing and consuming content, an evaluation of acceptance is part of future work.

- Another point is multimedia streaming. We assumed that the playback-buffer has the size of one unit, i.e., the full unit has to be downloaded to be played. Non-sequential media might require different buffer sizes than traditional multimedia streaming scenarios.

- We consider a homogeneous network of nodes, except for the scale-free network setup all nodes have the same role. One question is therefore, how the self-organizing algorithm adapts to heterogeneous capacities and capabilities of nodes. An overlay adaptation might be necessary.

- In the context of mobile systems energy consumption is an important topic. One could compare the energy efficiency of the proposed hormone-based algorithm with similar algorithms.

- The middleware should be applied to a number of different case studies to show its wide applicability. The middleware can be developed to a product by implementing it as a mobile application, e.g., on platforms like Android, iPhone, etc.

- Self-organizing algorithms are hard to apply because most of them need a precise configuration to work. In this thesis a genetic algorithm is used to find a fitting parameter set. However, there are some parameters for which typical settings exist, e.g., thanks to parameter studies from other researchers. To evaluate the effects of these parameters visualization could help. For the artificial hormone system such a visualization might contain the development of hormone concentrations in a network and the corresponding unit movements.

- In this thesis artificial hormone systems were used for task allocation and content delivery. Artificial hormone systems promise to be a new niche of bio-inspired algorithms. Although there are surveys on bio-inspired algorithms,

researchers that want to apply them have to rely on their experience to decide which algorithm to choose.

9. List of Publications

1. A. Sobe and L. Böszörmenyi, "Towards self-organizing multimedia delivery," Tech. Rep. 1, Klagenfurt University, TR/ITEC/12/2.08, 2008

2. A. Sobe and L. Böszörmenyi, "Non-sequential Multimedia Caching," in *2009 First International Conference on Advances in Multimedia (MMedia 2009)*, (Colmar, France), pp. 158–161, IARIA/IEEE, July 2009

3. A. Sobe, W. Elmenreich, and L. Böszörmenyi, "Towards a Self-organizing Replication Model for Non-sequential Media Access," in *Proceedings of the 2010 ACM workshop on Social, adaptive and personalized multimedia interaction and access (SAPMIA 2010)*, (Florence, Italy), pp. 3–8, ACM, 2010

4. A. Sobe, L. Böszörmenyi, and M. Taschwer, "Video Notation (ViNo): A Formalism for Describing and Evaluating Non-sequential Multimedia Access," *IARIA International Journal on Advances in Software*, vol. 3, no. 12, pp. 19–30, 2010

5. L. Böszörmenyi, M. del Fabro, M. Kogler, M. Lux, O. Marques, and A. Sobe, "Innovative Directions in Self-organized Distributed Multimedia Systems," *Multimedia Tools and Applications, Springer*, vol. 51, no. 2, pp. 525–553, 2010

6. A. Sobe, W. Elmenreich, and L. Böszörmenyi, "Replication for Bio-inspired Delivery in Unstructured Peer-to-Peer Networks," in *Ninth Workshop on Intellingent Solutions for Embedded Systems (WISES 2011)*, (Regensburg, Germany), p. 6, IEEE, 2011

7. A. Sobe, W. Elmenreich, and L. Böszörmenyi, "Storage Balancing in Self-organizing Multimedia Delivery Systems," *arxiv e-print 1111.0242; TR/ ITEC/ 01/ 2.13*, p. 16, Nov. 2011

8. A. Sobe, W. Elmenreich, and L. Böszörmenyi, "Artificial Hormone Systems as Middleware for Content Delivery," in *International Workshop on Self-Organizing Systems IWSOS 2012, submitted*, p. 12, 2012

A. ViNo EBNF Specification

The following ViNo specification [1] was created by using ANTLR a LL(*) parser generator [112]. Since special signs are not allowed in ANTLR, we used <_Q for $\leftarrow_Q$. This EBNF has been extended to allow the usage wildcard units. These units are marked by '?' and can be used like any other unit. Wildcard units can be specified by applying sequential operators in combination with further characteristics, e.g., QoS, tags, etc..

```
SEQ     : '<';
PAR     : '||';
NUMBER  :  ('0'..'9');
CHARS   :  ('a'..'z'|'A'..'Z');
STRING  :  (CHARS)+;
VALUE   :  (NUMBER)+;
NAME    :  '?'| CHARS ('.'|'-'|NUMBER|CHARS)*;
OP      :  '='|'>='|'<='|'<'|'>';

spec    : SEQ '_' (CHARS|STRING) OP (VALUE|NAME|NUMBER)
         (',' (CHARS|STRING) OP (VALUE|NAME|NUMBER))*;

unit    : NAME;

primitive : unit | group;

par     : (PAR primitive)+;

seq     : (SEQ primitive)+;

seqq    : (spec primitive)+;

pargroup : '[' primitive par ']';

seqgroup : '(' primitive (seq|seqq) ')';
```

```
group    :  pargroup | seqgroup;

comp     :( primitive ( par | seq| seqq )? )* EOF;
```

B. Interface Specification

During the work on the use cases of SOMA the interfaces to the composition/decomposition layer and the infrastructure layer have been defined.

Interface to the Application

In Appendix A the ViNo language specification has been defined. ViNo is the basis for the interface to an application. The corresponding ViNo parser implementation translates textual ViNo representations to Java and C# (Mono and Microsoft .Net).

To combine the delivery with the composition layer of SOMA, of a sample interface is defined in the following. The implementation is done in C# and to support inter-process communication .NET Remoting is used. The composition layer sends requests to the delivery layer and is notified if a requested unit is available for presentation, or if the request has been completed.

Thus, two interfaces are needed. First, one that allows sending requests and one that enables the callback. The middleware provides a method to receive a request. The client has to pass its callback Interface. The client identification allows the middleware to support several clients at once (proxy setting). In the following Figures B.1 and B.2 the interfaces are depicted.

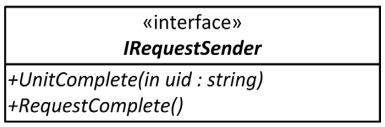


Figure B.1.: Interface for notifying the composition layer if a unit or a request is completed

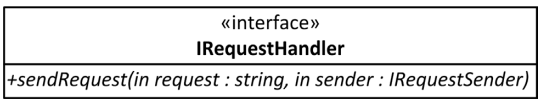


Figure B.2.: Request interface at the middleware to be used by the composition layer

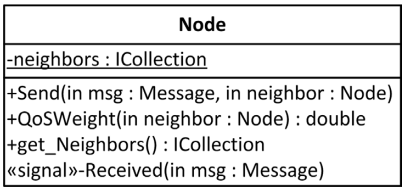


Figure B.3.: Node representation as interface to the network layer

Interface to the Network

The interface to the network has to provide the notion of a node. A node has to know its neighbors and should be able to send units and hormones to the neighbors. Furthermore, a notification mechanism is needed to indicate that a message arrived. The overlay building mechanism and peer churn handling should be transparent to the middleware.

The implemented basic network interface is shown in Figure B.3. The send method passes a Message object to a neighbor. A Message contains a list of hormones or a unit, i.e., the path to a physical file. For QoS-based delivery the QoS weight has to be calculated and stored in the collection (Dictionary) of neighbors. With the method `get_neighbors` all connections are updated and returned to the middleware. An event is thrown if a Message arrived.

Bibliography

[1] A. Sobe, L. Böszörmenyi, and M. Taschwer, "Video Notation (ViNo): A Formalism for Describing and Evaluating Non-sequential Multimedia Access," *IARIA International Journal on Advances in Software*, vol. 3, no. 12, pp. 19–30, 2010.

[2] A. Sobe, W. Elmenreich, and L. Böszörmenyi, "Towards a Self-organizing Replication Model for Non-sequential Media Access," in *Proceedings of the 2010 ACM workshop on Social, adaptive and personalized multimedia interaction and access (SAPMIA 2010)*, (Florence, Italy), pp. 3–8, ACM, 2010.

[3] L. Böszörmenyi, M. del Fabro, M. Kogler, M. Lux, O. Marques, and A. Sobe, "Innovative Directions in Self-organized Distributed Multimedia Systems," *Multimedia Tools and Applications, Springer*, vol. 51, no. 2, pp. 525–553, 2010.

[4] M. del Fabro, K. Schöffmann, and L. Böszörmenyi, "Instant Video Browsing: a Tool for fast Non-sequential Hierarchical Video Browsing," in *USAB 2010, HCI in Work and Learning, Life and Leisure, LNCS, vol. 6389/2010*, (Klagenfurt, Austria), pp. 443–446, Springer, 2010.

[5] M. del Fabro and L. Böszörmenyi, "The Vision of Crowds : Social Event Summarization Based on User-Generated Multimedia Content," in *ACM CHI 2011 Workshop Data Collection By The People For The People*, no. 1, (Vancouver, Canada), p. 3pp., ACM, 2011.

[6] A. Sobe and L. Böszörmenyi, "Non-sequential Multimedia Caching," in *2009 First International Conference on Advances in Multimedia (MMedia 2009)*, (Colmar, France), pp. 158–161, IARIA/IEEE, July 2009.

[7] A. Sobe, W. Elmenreich, and L. Böszörmenyi, "Replication for Bio-inspired Delivery in Unstructured Peer-to-Peer Networks," in *Ninth Workshop on Intellingent Solutions for Embedded Systems (WISES 2011)*, (Regensburg, Germany), p. 6, IEEE, 2011.

[8] A. Sobe, W. Elmenreich, and L. Böszörmenyi, "Storage Balancing in Self-organizing Multimedia Delivery Systems," *arxiv e-print 1111.0242; TR/ ITEC/ 01/ 2.13*, p. 16, Nov. 2011.

[9] D. Hausheer, P. Nikander, V. Fogliati, K. Wünstel, M. Callejo, S. Jorba, S. Spirou, L. Ladid, W. Kleinwächter, B. Stiller, M. Behrmann, M. Boniface, C. Courcoubetis, and L. Man-Sze, "Future Internet Socio-Economics - Challenges and Perspectives," in *Towards the Future Internet*, pp. 1–11, IOS Press, 2009.

[10] F. Heylighen, "The Science of Self-organization and Adaptivity," in *The Encyclopedia of Life Support Systems*, pp. 1–26, EOLSS.net, 2002.

[11] T. Klingberg and R. Manfredi, "Gnutella Protocol Specification 0.6: http://rfc-gnutella.sourceforge.net," 2002.

[12] F. Pletzer and B. Rinner, "Distributed Task Allocation for Visual Sensor Networks: A Market-based Approach," in *Workshop on Socio-Economics Inspiring Self-Managed Systems and Concepts (Seismyc)*, pp. 59–62, IEEE, 2010.

[13] M. del Fabro and L. Böszörmenyi, "Video Scene Detection Based On Recurring Motion Patterns," in *the Second International Conference on Advances in Multimedia, MMedia 2010, June 2010*, (Athens, Greece), IARIA/IEEE, 2010.

[14] S. Wieser and L. Böszörmenyi, "Flocks: Interest-Based Construction of Overlay Networks," in *2010 Second International Conferences on Advances in Multimedia (MMedia 2010)*, (Athens, Greece), pp. 119–124, IARIA/IEEE, 2010.

[15] M. Lux, M. Kogler, and M. del Fabro, "Why Did You Take this Photo: A Study on User Intentions in Digital Photo Productions," in *Proceedings of the 2010 ACM workshop on Social, adaptive and personalized multimedia interaction and access (SAPMIA 2010)*, (Florence, Italy), pp. 41–44, ACM, 2010.

[16] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Computing Surveys*, vol. 36, pp. 335–371, Dec. 2004.

[17] A. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, 2nd ed., 2006.

[18] J. Buford, H. Yu, and E. K. Lua, *P2P Networking and Applications*. Morgan Kaufmann, 2009.

[19] J. Li, "On Peer-to-Peer (P2P) Content Delivery," in *Peer-to-Peer Networking and Applications*, vol. 1, pp. 45–63, Springer, Jan. 2008.

[20] X. Li and J. Wu, "Searching Techniques in Peer-to-Peer Networks," in *Handbook of Theoretical and Algorithmic Aspects of Ad Hoc, Sensor, and Peer-to-Peer Networks*, pp. 613–642, Auerbach Publications, 2006.

[21] D. Tsoumakos and N. Roussopoulos, "Analysis and Comparison of P2P Search Methods," in *Proceedings of the 1st international conference on Scalable information systems - InfoScale '06*, (New York, USA), pp. 25–es, ACM, 2006.

[22] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and Replication in Unstructured Peer-to-Peer Networks," in *Proceedings of the 16th international conference on Supercomputing*, (New York, USA), pp. 84–95, ACM, 2002.

[23] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," in *Proceedings of the eleventh international conference on Information and knowledge management - CIKM '02*, (New York, New York, USA), p. 300, ACM, 2002.

[24] A. Crespo and H. Garcia-Molina, "Routing Indices for Peer-to-Peer Systems," in *Proceedings 22nd International Conference on Distributed Computing Systems (DCS)*, pp. 23–32, IEEE Computer Society, 2002.

[25] B. Yang and H. Garcia-Molina, "Efficient Search in Peer-to-Peer Networks," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, (Vienna, Austria), IEEE, 2002.

[26] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks," in *Proceedings Third International Conference on Peer-to-Peer Computing (P2P)*, pp. 102–109, IEEE Computer Society, 2003.

[27] T. Lin, P. Lin, H. Wang, and C. Chen, "Dynamic Search Algorithm in Unstructured Peer-to-Peer Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, pp. 654–666, May 2009.

[28] V. Cholvi, P. Felber, and E. Biersack, "Efficient Search in Unstructured Peer-to-Peer Networks," *European Transactions on Telecommunications*, vol. 15, pp. 535–548, Nov. 2004.

[29] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth Multicast in Cooperative Environments," in *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP)*, (New York, USA), pp. 298–313, ACM, 2003.

[30] V. Padmanabhan and K. Sripanidkulchai, "The Case for Cooperative Networking," *Peer-to-Peer Systems*, pp. 178–190, 2002.

[31] K. Mayer-Patel, "Systems Challenges of Media Collectives Supporting Media Collectives with Adaptive MDC," *Proceedings of the 15th international conference on Multimedia - MULTIMEDIA '07*, p. 625, 2007.

[32] M. Schiely and P. Felber, "CROSSFLUX: An Architecture for Peer-to-Peer Medias Streaming," *Emerging Communication*, vol. 8, pp. 342–358, 2006.

[33] B. Biskupski, M. Schiely, P. Felber, and R. Meier, "Tree-based Analysis of Mesh Overlays for Peer-to-Peer Streaming," in *Proc. of the 8th IFIP International conference on Distributed Applications and Interoperable Systems (DAIS '08)*, pp. 126–139, Springer, 2008.

[34] B. Cohen, "BitTorrent Protocol Specification (URL: http://www.bittorrent.org/beps/bep_0003.html)," 2003.

[35] B. Li and H. Yin, "Peer-to-Peer Live Video Streaming on the Internet: Issues, Existing Approaches, and Challenges," *IEEE Communications Magazine*, vol. 45, pp. 94–99, June 2007.

[36] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "BiToS: Enhancing BitTorrent for Supporting Streaming Applications," in *Proceedings 25TH IEEE International Conference on Computer Communications, (INFOCOM).*, pp. 1–6, IEEE, 2006.

[37] X. Hei, C. Liang, J. Liang, and Others, "Insights into PPLive: A Measurement Study of a Large-Scale P2P IPTV System," in *International Word Wide Web Conference (WWW). IPTV Workshop*, (Edinburgh, Scotland), 2006.

[38] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe, "Contracts: Practical Contribution Incentives for P2P Live Streaming," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pp. 6–6, USENIX Association, 2010.

[39] A. Begen, T. Akgul, and M. Baugher, "Watching Video over the Web: Part 1: Streaming Protocols," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 54–63, 2011.

[40] H. Schwarz and M. Wien, "The Scalable Video Coding Extension of the H.264/AVC Standard," *IEEE Signal Processing Magazine*, no. March, pp. 135–141, 2008.

[41] F. Heylighen and C. Gershenson, "The Meaning of Self-organization in Computing," in *IEEE Intelligent Systems*, pp. 72–75, 2003.

[42] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, *Self-organization in Biological Systems*. Princeton University Press, 2003.

[43] W. Ashby, "Principles of the Self-organizing System," *Principles of Self-organization*, vol. 6, no. 2000, pp. 255–278, 1962.

[44] W. Elmenreich, R. D'Souza, C. Bettstetter, H. de Meer, and H. D. Meer, "A Survey of Models and Design Methods for Self-organizing Networked Systems," in *Self-Organizing Systems: 4th IFIP TC 6 International Workshop, IWSOS 2009, Zurich, Switzerland, December 9-11, 2009, Proceedings*, p. 37, Springer-Verlag New York Inc, 2009.

[45] O. Babaoglu, A. Montresor, T. Urnes, G. Canright, A. Deutsch, G. a. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, and R. Montemanni, "Design Patterns from Biology for Distributed Computing," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 1, pp. 26–66, Sept. 2006.

[46] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli, "Case Studies for Self-organization in Computer Science," *Journal of Systems Architecture*, vol. 52, pp. 443–460, Aug. 2006.

[47] F. Dressler and O. B. Akan, "A Survey on Bio-Inspired Networking," *Computer Networks*, vol. 54, pp. 881–900, Apr. 2010.

[48] M. Dorigo, M. Birattari, and T. Stutzle, "Ant Colony Optimization," *IEEE Computational Intelligence Magazine*, vol. 1, pp. 28–39, Nov. 2006.

[49] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents.," *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics*, vol. 26, pp. 29–41, Jan. 1996.

[50] M. Dorigo and L. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53–66, Apr. 1997.

[51] G. D. Caro, "AntNet: Distributed Stigmergetic Control for Communications Networks," *Journal of Artificial Intelligence Research*, vol. 9, pp. 317–365, 1998.

[52] N. Franks and A. Sendova-Franks, "Brood Sorting by Ants: Distributing the Workload over the Work-surface," *Behavioral Ecology and Sociobiology*, vol. 30, pp. 109–123, Mar. 1992.

[53] C. Melhuish, O. Holland, and S. Hoddell, "Collective Sorting and Segregation in Robots with Minimal Sensing," in *5th International Conference on the Simulation of Adaptive Behaviour*, 1998.

[54] A. Khelil, C. Becker, J. Tian, and K. Rothermel, "An Epidemic Model for Information Diffusion in MANETs," *Proceedings of the 5th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems - MSWiM '02*, p. 54, 2002.

[55] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "Performance Modeling of Epidemic Routing," *Computer Networks*, vol. 51, pp. 2867–2891, July 2007.

[56] J. Holland, *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[57] M. Srinivas and L. Patnaik, "Genetic Algorithms: A Survey," *Computer*, vol. 27, pp. 17–26, June 1994.

[58] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor, "Firefly-inspired Heartbeat Synchronization in Overlay Networks," in *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pp. 77–86, IEEE, July 2007.

[59] D. Dasgupta, "Advances in Artificial Immune Systems," *IEEE Computational Intelligence Magazine*, vol. 1, pp. 40–49, Nov. 2006.

[60] L. De Castro and F. Von Zuben, "Artificial Immune Systems: Part I - Basic Theory and Applications," tech. rep., Universidade Estadual de Campinas, 1999.

[61] A. Forestiero, C. Mastroianni, and G. Spezzano, "Building a Peer-to-peer Information System in Grids via Self-organizing Agents," *2007 2nd Bio-Inspired Models of Network, Information and Computing Systems*, vol. 6, pp. 125–140, Dec. 2007.

[62] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Computer Vision, 2009 IEEE 12th International Conference on*, no. Iccv, pp. 2130–2137, IEEE, 2009.

[63] A. Forestiero, C. Mastroianni, and G. Spezzano, "QoS-based Dissemination of Content in Grids," *Future Generation Computer Systems*, vol. 24, pp. 235–244, Mar. 2008.

[64] E. Michlmayr, *Ant algorithms for self-organization in social networks*. PhD thesis, Technical University Vienna, Austria, 2007.

[65] K.-H. Yang, C.-J. Wu, and J.-M. Ho, "AntSearch: An Ant Search Algorithm in Unstructured Peer-to-Peer Networks," *11th IEEE Symposium on Computers and Communications (ISCC'06)*, no. 1, pp. 429–434, 2007.

[66] N. Ganguly, L. Brusch, and A. Deutsch, "Design and Analysis of a Bio-inspired Search Algorithm for Peer-to-Peer Networks," in *Self-Star Properties in Complex Information Systems*, pp. 358–372, Springer, 2005.

[67] M. Lux, O. Marques, K. Schöffmann, L. Böszörmenyi, and G. Lajtai, "A Novel Tool for Summarization of Arthroscopic Videos," *Multimedia Tools and Applications*, vol. 46, pp. 521–544, Sept. 2009.

[68] D. Bulterman and L. Rutledge, *SMIL 3.0: Flexible Multimedia for Web, Mobile Devices and Daisy Talking Books*. Springer, 2008.

[69] K. Nahrstedt and J. Jin, "QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy," *Multimedia, IEEE*, vol. 11, pp. 74–87, July 2004.

[70] J. Altmann and P. Varaiya, "INDEX Project: User Support for Buying QoS with Regard to User's Preferences," in *Sixth International Workshop on Quality of Service (IWQoS'98)*, pp. 101–104, IEEE, 1998.

[71] X. Gu, "An XML-based Quality of Service Enabling Language for the Web," *Journal of Visual Languages & Computing*, vol. 13, pp. 61–95, Feb. 2002.

[72] S. Frolund and J. Koistinen, "Qml: A Language for Quality of Service Specification," tech. rep., HP Technical Reports HPL-98-10, 1998.

[73] I. Foster and C. Kesselman, "The Globus Project: A Status report," in *Proceedings Seventh Heterogeneous Computing Workshop (HCW'98)*, pp. 4–18, IEEE Comput. Soc, 1995.

[74] S. Shenker, R. Braden, and D. Clark, "Integrated Services in the Internet Architecture: An Overview," *IETF Request for Comments (RFC)*, vol. 1633, 1994.

[75] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Service," *IETF Request for Comments (RFC)*, vol. 2475, 1998.

[76] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol:(RSVP)," *IETF Request for Comments (RFC)*, vol. 2205, 1997.

[77] G. S. Blair and J.-B. Stefani, *Open Distributed Processing and Multimedia*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[78] O. Lampl, E. Stellnberger, and L. Böszörményi, "Programming Language Concepts for Multimedia Application Development," *Modular Programming Languages*, vol. 4228, pp. 23–36, 2006.

[79] O. Lampl and L. Böszörmenyi, "Adaptive Quality-Aware Programming with Declarative QoS Constraints," in *IASTED International Conference on Internet and Multimedia Systems and Applications* (M. Roccetti, ed.), IASTED, 2008.

[80] J. Le Boudec and P. Thiran, "Network Calculus: A Theory of Deterministic Queuing Systems for the Internet," *LNCS*, vol. 2050, p. 274p., 2001.

[81] D. Pandit, *Quality of Service Performance Analysis based on Network Calculus*. PhD thesis, Technische Universität Darmstadt, Germany, 2006.

[82] G. Pallis and A. Vakali, "Insight and Perspectives for Content Delivery Networks," *Communications of the ACM*, vol. 9, no. 1, pp. 101–106, 2006.

[83] A. Vakali and G. Pallis, "Content Delivery Networks: Status and Trends," *Internet Computing, IEEE*, vol. 7, no. 6, pp. 68–74, 2003.

[84] K. Stamos, G. Pallis, A. Vakali, D. Katsaros, A. Sidiropoulos, and Y. Manolopoulos, "CDNsim: A Simulation Tool for Content Distribution Networks," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 20, no. 2, pp. 1–40, 2010.

[85] A. Varga, "OMNeT++," in *Modeling and Tools for Network Simulation* (K. Wehrle, M. Günes, and J. Gross, eds.), pp. 35–59, Springer Berlin / Heidelberg, 2010.

[86] S. Podlipnig and L. Böszörmenyi, "A Survey of Web Cache Replacement Strategies," *ACM Computing Surveys*, vol. 35, pp. 331–373, Dec. 2003.

[87] J. Yu, C. T. Chou, Z. Yang, X. Du, and T. Wang, "A Dynamic Caching Algorithm based on Internal Popularity Distribution of Streaming Media," *Multimedia Systems*, vol. 12, pp. 135–149, July 2006.

[88] S. Chen, B. Shen, S. Wee, and X. Zhang, "Adaptive and Lazy Segmentation based Proxy Caching for Streaming Media Delivery," *Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video - NOSSDAV '03*, p. 22, 2003.

[89] L. Guo, S. Chen, Z. Xiao, X. Zhang, P. Ave, and F. Park, "DISC: Dynamic Interleaved Segment Caching for Interactive Streaming," *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pp. 763–772, 2005.

[90] Y. Zhao, D. L. Eager, and M. K. Vernon, "Scalable On-Demand Streaming of Non-linear Media," *IEEE/ACM Transactions on Networking*, vol. 15, pp. 1149–1162, Oct. 2007.

[91] C. Kofler and M. Lux, "Dynamic Presentation Adaptation based on User Intent Classification," in *Proceedings of the seventeenth ACM international conference on Multimedia*, (New York, New York, USA), pp. 1117–1118, ACM, 2009.

[92] W. Tang, Y. Fu, and L. Cherkasova, "Medisyn: A Synthetic Streaming Media Service Workload Generator," in *Proceedings of the 13th international workshop on Network*

and operating systems support for digital audio and video - NOSSDAV '03, pp. 12–21, ACM, 2003.

[93] L. Rushton, *The Endocrine System.* Chelsea House, 2004.

[94] U. Brinkschulte, M. Pacher, and A. Von Renteln, "Towards an Artificial Hormone System for Self-organizing Real-time Task Allocation," *Software Technologies for Embedded and Ubiquitous Systems*, pp. 339–347, 2007.

[95] L. Rong, "Multimedia Resource Replication Strategy for a Pervasive Peer-to-Peer Environment," *Journal of Computers*, vol. 3, pp. 9–15, Apr. 2008.

[96] K. Herrmann, "Self-organizing Replica Placement - A Case Study on Emergence," *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pp. 13–22, July 2007.

[97] F. Heylighen, "Evolution, Selfishness and Cooperation," *Journal of Ideas*, vol. 2, no. 4, pp. 70–76, 1992.

[98] M. Satyanarayanan, V. Bahl, and R. Caceres, "The Case for VM-based Cloudlets in Mobile Computing," *Pervasive Computing*, 2009.

[99] A. Sobe, W. Elmenreich, and L. Böszörmenyi, "Artificial Hormone Systems as Middleware for Content Delivery," in *International Workshop on Self-Organizing Systems IWSOS 2012, submitted*, p. 12, 2012.

[100] W. Elmenreich and G. Klingler, "Genetic Evolution of a Neural Network for the Autonomous Control of a Four-wheeled Robot," in *Sixth Mexican International Conference on Artificial Intelligence, Special Session*, pp. 396–406, IEEE, 2007.

[101] H. Yamamoto, D. Maruta, and Y. Oie, "Replication Methods for Load Balancing on Distributed Storages in P2P Networks," *The 2005 Symposium on Applications and the Internet*, pp. 264–271, 2005.

[102] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," in *Designing Privacy Enhancing Technologies*, pp. 46–66, Springer, 2001.

[103] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-Peer Networks," in *ACM SIGCOMM Computer Communication Review*, vol. 32, pp. 177–190, ACM, Oct. 2002.

[104] E. Leontiadis, V. Dimakopoulos, and E. Pitoura, "Creating and Maintaining Replicas in Unstructured Peer-to-Peer Systems," in *Euro-Par 2006 Parallel Processing*, pp. 1015–1025, Springer, 2006.

[105] D. Eppstein and J. Wang, "A Steady State Model for Graph Power Laws," *Arxiv e-print cs/0204001*, 2002.

[106] A. Renteln and U. Brinkschulte, "The Artificial Hormone System - An Organic Middleware for Self-organizing Real-time Task Allocation," in *Organic Computing – A Paradigm Shift for Complex Systems*, ch. 4.4, pp. 369–384, Springer Berlin / Heidelberg, 2011.

[107] S. Balasubramaniam, D. Botvich, J. Mineraud, W. Donnelly, and N. Agoulmine, "BiRSM: Bio-inspired Resource Self-management for all IP-networks," *Network, IEEE*, vol. 24, no. 3, pp. 20–25, 2010.

[108] G. Tempesti, F. Vannel, P. Mudry, and D. Mange, "A Novel Platform for Complex Bio-inspired Architectures," in *IEEE Workshop on Evolvable and Adaptive Hardware, 2007. WEAH 2007.*, pp. 8–14, IEEE, 2007.

[109] P. Harsh, R. Chow, and R. Newman, "Gray Networking: A Step towards Next Generation Computer Networks," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 1323–1328, ACM, 2010.

[110] P. Marciniak, N. Liogkas, A. Legout, and E. Kohler, "Small is not Always Beautiful," in *Proceedings of the 7th international conference on Peer-to-peer systems*, pp. 9–9, USENIX Association, 2008.

[111] A. Sobe and L. Böszörmenyi, "Towards self-organizing multimedia delivery," Tech. Rep. 1, Klagenfurt University, TR/ITEC/12/2.08, 2008.

[112] T. Parr, *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 2007.