

# An Architecture for Complex P2P Systems

Sebastian Holzapfel, Arno Wacker, Torben Weis and Matthäus Wander

University of Duisburg-Essen, Distributed Systems Group,  
Bismarckstrasse 90, 47057 Duisburg, Germany  
{sebastian.holzapfel|arno.wacker|torben.weis|matthaeus.wander}@uni-due.de

**Abstract**—This article presents an architecture for research and development of peer-to-peer (P2P) systems. A complete P2P application has to cope with problems such as NAT-traversal, bootstrapping, connection management, routing, storage, and security. Therefore, our approach separates the system into layers and components. A developer can easily build a complete P2P stack by plugging layers and components together, which allows for easy code reuse and interchangeability. Furthermore, our architecture allows us to run a discrete event simulation by using a special programming model. This way we can use the same code base for productive applications as well as for measurements & tests on a compute cluster. Our evaluation shows that using our architecture has a negligible effect on performance and a very small memory footprint, which allows us to simulate thousands of peer instances running the real application code on a single machine.

**Index Terms**—Peer-to-Peer, P2P, architecture, protocol stack.

## I. INTRODUCTION

During the last years we have been working on a P2P-based massively multiplayer game framework named peers@play (p@p) [1]. The development of p@p has shown that building complex P2P applications requires more than forming an overlay structure and a routing algorithm. Although the overlay is often the main focus of scientific interest, other issues such as bootstrapping, NAT-traversal, and connection management are non-trivial tasks. Consequently, it is significantly less work to simulate an overlay than to run real-world experiments where the software is really deployed on nodes behind NATs using DSL connections with varying bandwidth, and latency.

We realized that these real-world influences can make all the difference between a game that is playable or a game that is too slow or too unstable. For example, some 100ms to perform UDP hole punching [2] is significant when the user expects a response in 100-150ms. We concluded that real-world tests are a must to demonstrate the feasibility of such applications, especially when they are as sensitive to latency, availability, and security as P2P online games. Especially when developing towards a productive application to be shipped eventually, real-world tests must not be omitted. Despite the need of real-world testing, there is still a need to perform lab experiments with new protocols. The common approach is to simulate the protocols for this purpose. In network simulators the software under test is often rewritten to comply with the simulation framework. Hence, two implementations for the same concept are required: one version for simulation purposes and one for real-world tests. To circumvent this, the first requirement (R1)

for our approach was that we wanted to use a single code base for measurements on a compute cluster and for real-world testing. Thereby, it is important that the developed solution causes a low overhead. Otherwise, it would be unfeasible to simulate thousands of peers.

Furthermore, we wanted to reuse solutions developed in one project (for example bootstrapping protocols [3], NAT-traversal [4], or routing protocols [5], [6]) in several research projects and to evaluate our system using different solutions. Thus, we aimed at a system of pluggable components. This allows us, for example, to develop an experimental overlay as one component and to plug it together with other already existing components that perform more basic services such as bootstrapping and connection management, or more high-level services such as storage or an example application. The major challenge is to ensure the interoperability of the different implementations. Thus, the second requirement (R2) for our system is code reuse and interchangeability.

In p@p we are using multiple overlays for different purposes. For example, a P2P storage must provide information about the whereabouts of players and goods in the 2- or 3-dimensional virtual world. Thus, the storage must support multi-dimensional range queries as offered by e.g. a Voronoi overlay [7]. In contrast to that, storing player characters (e.g. skill levels, achievements & equipment) requires heavy replication to achieve optimal persistency of data. For these purposes a reliable distributed hash table (DHT) is better suited. Our architecture allows multiple overlay instances to share the same lower layers. Thereby, it is important no overlay starves another overlay, because otherwise the whole system performance will be decreased. Hence, our third requirement (R3) is to support multi-protocol scenarios.

Summarizing our architecture has to fulfill the following requirements:

- R1: Single code base for simulation and release software introducing low overhead**
- R2: Code reuse and interchangeability**
- R3: Multi-protocol support**

The paper is structured as follows: In Section II we describe our architecture. After that we evaluate in Section III the performance impact of our architecture and discuss the memory footprint. Finally, we discuss related work and close the paper with concluding remarks and an outlook on further work.

## II. OUR APPROACH

While working on various protocols and demonstrators for the p@p project, we realized that p@p became much more complex than typical P2P applications. The main drivers for complexity were multi-protocol support, testing and measurement support, as well as issues related to NAT-traversal, bootstrapping, etc. Following established principles in distributed systems design, we chose a layered P2P architecture building on top of TCP/IP (Figure 1).

Similar to other P2P architectures, e.g. as proposed by Aberer [8], our architecture includes an application, a storage and an overlay layer. Our application layer covers, besides the actual application logic and integration with the visualization, reusable components for typically recurring P2P problems like *Group Communication* and *Election*. The storage layer provides a DHT-like key-value interface to store and retrieve data in and from the P2P network and ensures reliability by using replication. The overlay layer is responsible to form a structure of interconnected peers and to route messages in this structure between peers. In our overlay layer multiple overlays can be used concurrently for different purposes, e.g. a Voronoi overlay [7] to exchange location-based data of a virtual environment and a Chord [9] overlay for data storage.

Our major difference to other P2P architectures is the P2P link layer. This layer, not to be confused with the TCP/IP link layer, is built directly on top of the TCP/UDP transport and consists of two components. The first one, bootstrapping, offers functionality to find existing nodes of an overlay network and to join a network. For this purpose we can use e.g. our IRC-based bootstrapping as described in [3]. The second component, P2P link management, is used to establish direct connections between peers in presence of NAT-based routers. Due to the fact that NAT-traversal is complex and costly, it is favorable to share already established links between concurrently running overlays. Thus, an instance of the P2P link management is shared between multiple overlays and uses message multiplexing. We will describe the P2P link management in detail in Section II-A.

Our architecture uses a dedicated scheduler that controls the execution of all components and a timer object that is responsible to trigger timed events, intervals and timeouts. We will explain the purpose of these components in Section II-B.

### A. P2P Link Management

The purpose of the P2P link management is to handle direct communication between two peers based on TCP or UDP transport. Indirect communication by routing messages between peers is not part of the P2P link management, although the overlay layer relies on it to send a message point-to-point to the next peer en route.

A peer that is connected to the Internet via a NAT-router has two IP addresses: an internal one being used in the local area network, e.g. 192.168.x.x., and an external one used by the NAT-router for outward communication. The port number assigned by the NAT-router may also differ from the internal one used by the P2P application, thus leading to two distinct

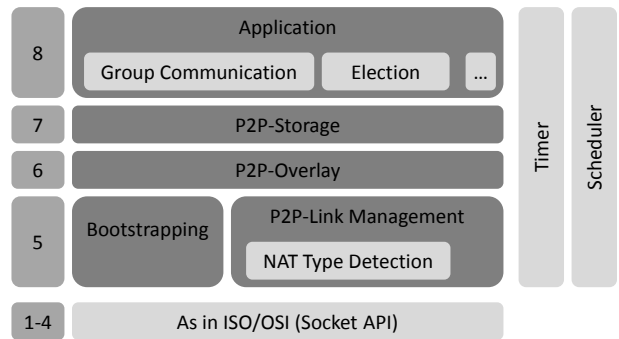


Fig. 1. p@p Architecture

IP addresses and port numbers. To deal with this circumstance the P2P link management uses a custom addressing scheme. A P2P link address is a tuple of the form

(*internal IP addr./port, external IP addr./port, NAT type*).

If two peers happen to have the same external IP address, then our P2P link management tries to connect to the internal IP endpoint first. The reason to bypass the NAT-router is, apart from a performance benefit, that some NAT-routers do not support hair-pinning [2] and thus communication via the external IP address is not possible if both peers are on the same local network.

If the external IP address differs or the internal connection fails, the P2P link management uses the external IP endpoint as destination. However, the success depends on the NAT type of both peers involved [10]. If at least one peer is not behind a NAT-router, a direct connection or connection reversal can be used. If both peers are behind a NAT-router, hole punching may work, depending on the mapping and filtering behavior of the NAT-routers, else we have to resort to relaying. Thus, a peer attempting to establish a P2P link to another one considers both, its own and the destination NAT type. To determine the NAT type of a peer we developed a protocol named MFB as described in [11].

Two peers trying to establish a direct connection through NAT-routers, e.g. by UDP hole punching [2], must somehow negotiate connection parameters and coordinate their actions. Since they are not yet connected, we rely on the overlay to route message between the two peers, assuming that they both have already joined a common overlay. Thus, the P2P link management calls the above overlay layer to route a message between peers. While this is not the usual data flow in a layered network architecture, it enables us to improve the connectivity of the P2P link layer by reusing already initialized and interconnected components.

A similar situation occurs when a directly reachable peer without any NAT or packet filtering is needed to relay messages between restricted peers. In our architecture we rely on the P2P storage to save addresses of suitable relaying peers. Thus, the P2P link management can query the above storage layer to retrieve addresses of relaying peers.

As multiple overlays use the same instance of the P2P link

management, they share already established P2P links. For example, if two peers are neighbors in a Voronoi overlay [7], they can potentially put each other in their Chord [9] routing table and benefit from closer routing paths. This way we can reduce the overall number of P2P links and thus save some cost on connection establishment.

The P2P link management performs a special link garbage collection that diverges from the traditional open/close semantics of sockets and file handles. If a link is idle for some time and if the number of links exceeds a threshold then the P2P link management queries all overlays whether the link is still needed. An overlay can either insist on the existence of a link (usually if it is a direct neighbor in the overlay structure), or it can remain indifferent. The link is kept alive, if at least one overlay insists on keeping it open, otherwise it is closed and the overlays are informed. Thus, overlays never explicitly close a link, but just have to distinguish between essential links and links which may be used if they happen to exist.

As the P2P link management uses multiplexing to send messages from multiple overlays over a single TCP/UDP transport, this opens the problem of queuing messages. For example, a large file transfer for distributing new content in a virtual environment will force the P2P link management into queuing and lead to delay of other messages. Our experience has shown, that delays of overlay maintenance messages can trigger timeouts and unnecessarily decrease the quality of service of the P2P network. A file transfer can easily smash a ring overlay by delaying maintenance messages. While this could be tackled by dynamically adjusting timeouts, the file transfer rendered our P2P game unplayable, because the delay was too high. Consequently, we implemented priority scheduling in the P2P link management, similar to the algorithm available for DiffServ [12]. We are using the following priority classes:

- High: Messages related to real-time user interaction
- Medium: Messages related to overlay maintenance
- Low: File transfer

The scheduler sends messages with high priority first. However, we must avoid starvation of messages with medium priority since this could damage the overlay. Thus, if a medium priority message is queued too long, its priority is upgraded to avoid starvation. Messages of low priority are only sent when possible. Their priority is never upgraded.

However, this scheduling may still cause the upper layers (which are responsible for file transfers) to retransmit low priority messages that are queued for too long. As a result we introduced an interface in the P2P link management to determine the current queue length. Thus, a well-behaved upper layer sending low priority messages periodically checks the queue length to adjust its flow.

### B. Scheduler, Timer and Socket API

One requirement of our architecture is that we can use the same code base for simulation and release software (R1). Thereby it is important that we are able to support a high number of peer instances on a single computer. Hence, we require a system which carefully households the available resources like

threads and memory. To achieve this, we designed Gears4Net (G4N) in [13], an asynchronous programming model for scalable and distributed applications. The main entity in G4N is a *protocol* and multiple protocols communicate with each other by passing messages through message queues. Each protocol requires a *scheduler* to coordinate the execution inside the protocol, i.e. whenever some part of the protocol exits a waiting state and is therefore ready to perform some work, the scheduler assigns a worker thread. A scheduler can use one or more worker threads and different protocols can share the same scheduler. Hence, G4N allows for executing concurrently any number of protocols with only a fixed number threads, e.g. as many as CPU cores are available.

To run a discrete event simulation we need to control the execution time as well as the time behavior. To do so, we use a single simulation scheduler and a timer object for all running instances. Thereby, the timer object uses a virtual time instead of real time and the simulation scheduler controls the execution of each instance.

Additionally, the P2P link management has to use virtual sockets instead of real network socket. Thereby, all message are exchanged using local message passing. In order to make the simulation more realistic, the virtual socket are able to simulate varying bandwidth and latency.

## III. EVALUATION

In this section, we evaluate our architecture in terms of processing and memory overhead. All layers, components and applications we developed for this evaluation have been written using Microsoft .NET C# 4.0. They have been executed on a single Lenovo ThinkPad T61p with a 2.6Ghz processor, 3GB RAM and the Windows 7 / 32-bit operating system.

### A. Processing Time

In the following, we evaluate the overhead in terms of processing time caused by using our architecture and G4N. We do this by measuring the round-trip time of a message in various settings. The message thereby travels through all layers downwards when sent and upwards when being received.

In the first setting, we implemented two simple applications to measure the overhead incurred solely by separating layers via G4N. Application (A) exchanges all messages immediately without layering and G4N by using standard TCP sockets. Application (B) in addition uses three layers and G4N. Each layer thereby simply forwards the message to the next layer without processing any further application logic. By subtracting the time required for (A) from (B) we can estimate the overhead of layering. Using both applications (A) and (B), we measured the round-trip time 1,000,000 times. Each measurement consists of two messages. The request message has a size of 290 bytes and the reply message has a size of 258 bytes. The results are shown in Table I. The columns  $Q1$  and  $Q3$  correspond to the lower and upper quartile, respectively, between which 50% of all values are situated. By subtracting from (B) the time required by (A), we can identify the overhead incurred by three layers on each peer, which is

0.02ms. When considering an average network latency in the Internet of between 30ms to 40ms, this overhead is negligible.

In the second setup, we measured the round-trip time by using the p@p framework (Application C). Thereby we used a DHT based on a Chord overlay as well the P2P link management. All layers and components are fully functional in this setup. We issue a STORE request on one peer that is being processed by the other peer. We measure the time until the peer receives the acknowledgement for its STORE request. To ensure the comparability of results obtained from applications (A) and (B), the message sizes in both settings are equal. In accordance to the first setup, we measured the round-trip time of 1,000,000 STORE operations. For (C) the round-trip is 0.65ms as shown in Table I, too.

In summary, our performance measurements show that time spent on layering is almost an order of magnitude smaller than the time required for sending a message via TCP on the loopback device (compare applications (A) and (B)). The cost of the real application including the whole application logic is yet another order of magnitude higher than the cost of plain TCP communication. This clearly highlights that the application and networking logic is costly in terms of performance and demonstrates that our architecture yields a high flexibility with no significant performance impact.

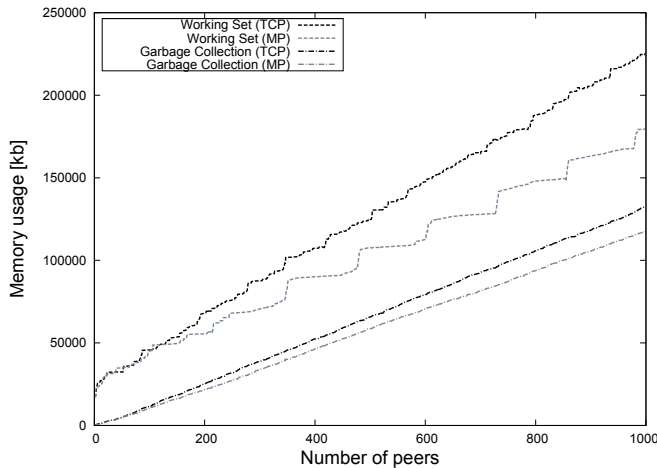


Fig. 2. 1000 connected peers (Chord) with a total of 5190 connections

	Min	Max	Q1	Q2 (Median)	Q3	Average
A	0.098	0.130	0.099	0.102	0.109	0.105
B	0.108	0.147	0.114	0.120	0.126	0.121
C	0.648	0.752	0.656	0.659	0.665	0.663

TABLE I  
AVERAGE ROUND-TRIP TIME IN MILLISECONDS

## B. Memory

In this section we investigate the number of peers that can be emulated on a single machine with the p@p framework. Therefore, we instantiate as many peers as possible in a single process. Since G4N can schedule any number of protocols

on a fixed number of threads (e.g. one thread for each CPU core), memory for the heap is the only restricted resource. We measured the memory allocation for a varying number of peers. The memory usage has been recorded for a P2P network using Chord [9] as overlay protocol.

The measurement, illustrated in Figure 2, includes the memory usage of both, in-process message passing and TCP transport protocols. Since TCP requires more memory for buffering than in-process message passing, the memory usage of using TCP is higher than that of in-process message passing. This measurement underpins the importance of replacing the network sockets with simulation sockets since it allows us to instantiate more peers on a single machine.

In summary, we are able to run a large amount of peers on a standard PC and main memory is the only restricted resource. It is important to notice that all layers and components were running productivity code.

## IV. RELATED WORK

Various network and P2P simulators exist to simulate an underlay or P2P network. In fact, NS-2 [14] is a discrete-event-simulator, suitable for simulating packet switching and small scale networks. However, it focuses only on the ISO/OSI layers 1 to 4 rather than on upper layers. In contrast, P2P simulators like PeerSim [15], P2PRealm [16], PlanetSim [17] are able to simulate a complete P2P network. However, both network and P2P simulators use a clear separation between simulation and real-world code, thus failing to meet *R1*.

In contrast to the discussed simulator, development tools like Neko [18], MACEDON [19], and RealPeer [20] allow for reusing the code base. However, since Neko [18] is tailored towards smaller distributed systems, it does not support thousands of nodes as required for P2P simulations. MACEDON [19] is an infrastructure to support the design, development and evaluation of P2P overlays. Its usability is limited by the fact that it only supports DHTs and application level multicast. In [20] the authors proposed RealPeer, a methodology and framework for the simulation-based development of P2P systems. It supports simulations and real-world applications by iteratively transforming a model of the P2P system into the real one. However, it lacks of multi-protocol support.

A multitude of reference architectures for P2P systems exist, such as Aberer [8], Dabek [21] and Lua [22]. They all describe a layering architecture without detailed information. Most of these architectures distinguish between an overlay and storage layer, but none of these architectures consists of a layer covering the functionalities provided by the P2P link layer. In addition to this, none of these architectures include functionalities to use the same code base for simulation and the productive application.

P2P Frameworks like JXTA [23], SpoVNet [24], and OPeN [25] do not support multi-protocol scenarios. They further do not support simulation using the productive code so that they fail to meet *R1*.

## V. CONCLUSION AND FUTURE WORK

In this paper, we detailed our P2P architecture, driven by numerous practical experiences and lessons learned. Our architecture is applicable for simulations as well as for real-world applications and fulfills all requirements. It is the foundation of our p@p framework, which has already been used in two real-world applications: a demo prototype of a purely P2P-based virtual 3D environment at CeBit'2010.

By using our asynchronous programming model G4N and special simulation components we are able to use the same code base for the productive application as well as for simulations and measurements. We evaluated our approach with respect to memory and latency overhead in Section III. The results show, that both the layering and the enhancements deduced from our requirements, come with an acceptable overhead, so that R1 is fulfilled. By separating the functionality in individual layers, we can substitute implementations for easy code reuse and interchangeability (R2). The introduction of the P2P link layer allows us to run multiple overlay concurrently, so that R3 is fulfilled.

In the future, we plan to adapt our systems at runtime by exchanging layers on the fly. This would allow us to exchange overlays depending on the network size. This is possible because our layers communicate only by exchanging messages asynchronously. Thus, we can halt the message queue, insert a different layer implementation and resume the queues.

## REFERENCES

- [1] "Peers@Play," Published on the WWW at <http://www.peers-at-play.org>, (retrieved on 2011-07-01).
- [2] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-Peer Communication Across Network Address Translators," in *USENIX Annual Technical Conference*, 2005, pp. 179–192.
- [3] M. Knoll, A. Wacker, G. Schiele, and T. Weis, "Bootstrapping in Peer-to-Peer Systems," in *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*, Melbourne, Australia, 2008.
- [4] A. Wacker, G. Schiele, S. Holzapfel, and T. Weis, "A NAT Traversal Mechanism for Peer-To-Peer Networks," in *P2P '08: Proceedings of the Eighth IEEE International Conference on Peer-to-Peer Computing (P2P'08)*. Aachen, Germany: IEEE, Sep. 2008, pp. 81–83.
- [5] M. Saternus, M. Knoll, F. Dürr, and T. Weis, "Symstry: Ein P2P-System für Ortsbezogene Anwendungen," in *Proceedings of 15. ITG/GI - Fachtagung (KiVS 2007)*. VDE-Verlag, Februar 2007, Konferenz-Beitrag, pp. 99–104.
- [6] M. Knoll and T. Weis, "A P2P-Framework for Context-based Information," in *1st International Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI) at Pervasive 2006*, Dublin, Ireland, May 2006.
- [7] O. Beaumont, A.-M. Kermarrec, L. Marchal, and E. Riviere, "VoroNet: A scalable object network based on Voronoi tessellations," in *Proc. IEEE Intl. Parallel and Distributed Computing Symp. (21st IPDPS'07)*. Long Beach, California, USA: IEEE Computer Society (Los Alamitos, CA), Mar. 2007, pp. 1–10.
- [8] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth, "The Essence of P2P: A Reference Architecture for Overlay Networks," in *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 11–20.
- [9] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications," in *Proceedings of the 2001 ACM SIGCOMM Conference*. San Diego, California, United States: ACM, 2001, pp. 149–160.
- [10] S. Guha and P. Francis, "Characterization and Measurement of TCP Traversal through NATs and Firewalls," *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, 2005.
- [11] S. Holzapfel, M. Wander, A. Wacker, L. Schwittmann, and T. Weis, "A New Protocol to Determine the NAT Characteristics of a Host," in *Proceedings of 25th IEEE International Parallel Distributed Processing Symposium, International Workshop on Hot Topics in Peer-to-Peer Systems (HOTP2P)*, Anchorage, Alaska, USA, May 2011.
- [12] K. Nichols, S. Blake, F. Baker, and D. Black, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, IETF RFC2474, 1998.
- [13] M. Saternus, T. Weis, S. Holzapfel, and A. Wacker, "Gears4Net An Asynchronous Programming Model," in *International Conference on Parallel Processing Workshops, 2009. ICPPW '09.*, Sep. 2009.
- [14] "The Network Simulator - ns-2," <http://www.isi.edu/nsnam/ns/>, (retrieved on 2011-07-01).
- [15] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Peer-to-Peer Computing, 2009. P2P '09. IEEE Ninth International Conference on*, Sep. 2009, pp. 99–100.
- [16] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, and J. Vuori, "P2PRealm – Peer-to-Peer Network Simulator," in *Proc. 11th International Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks, 2006, 2006*, pp. 93–99.
- [17] P. G. López, C. Pairet, R. Mondéjar, J. P. Ahulló, H. Tejedor, and R. Rallo, "PlanetSim: A New Overlay Network Simulation Framework," in *SEM*, vol. 3437. Springer, 2004, pp. 123–136.
- [18] P. Urbán, X. Défago, and A. Schiper, "Neko: A Single Environment to Simulate and Prototype Distributed Algorithms," in *Proceedings of the 15th Int'l Conf. on Information Networking (ICOIN-15)*, Beppu City, Japan, 2001.
- [19] A. Rodriguez, C. Killian, S. Bhat, D. Kotic, and A. Vahdat, "MACE-DON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks," in *In NSDI*, 2004, pp. 267–280.
- [20] D. Hildebrandt, L. Bischofs, and W. Hasselbring, "RealPeer-A Framework for Simulation-Based Development of Peer-to-Peer Systems," in *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, ser. PDP '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 490–497.
- [21] F. Dabek, B. Zhao, P. Druschel, J. Kubiawicz, and I. Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays," in *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*. Springer, 2003.
- [22] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Surveys and Tutorials*, vol. 7, pp. 72–93, 2005.
- [23] L. Gong, "JXTA: A Network Programming Environment," *IEEE Internet Computing*, vol. 5, pp. 88–95, 2001.
- [24] R. Bless, C. Huebsch, S. Mies, and O. Waldhorst, "The Underlay Abstraction in the Spontaneous Virtual Networks (SpoVNet) Architecture," *Next Generation Internet Networks, 2008. NGI 2008*, apr. 2008.
- [25] E. Tanin, A. Harwood, H. Samet, S. Nutanong, and M. T. Truong, "A Serverless 3D World," in *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*. New York, NY, USA: ACM, 2004, pp. 157–165.