Orbweb—A Network Substrate for Peer-to-Peer Desktop Grid Computing Based on Open Standards

Sven Schulz, Wolfgang Blochinger and Mathias Poths

```
@Article{Schulz2010a,
	author="Schulz, Sven and Blochinger, Wolfgang and Poths, Mathias",
	title="Orbweb---A Network Substrate for Peer-to-Peer Desktop Grid Computing
	Based on Open Standards",
	journal="Journal of Grid Computing",
	year="2010",
	month="Mar",
	day="01",
	volume="8",
	number="1",
	pages="77--107",
	issn="1572-9184",
	doi="10.1007/s10723-009-9121-8",
	url="https://doi.org/10.1007/s10723-009-9121-8"
}
```

Journal of Grid Computing manuscript No.
(will be inserted by the editor)

Orbweb – A Network Substrate for Peer-to-Peer Grid Computing based on Open Standards.

Sven Schulz · Wolfgang Blochinger · Mathias Poths

Received: date / Accepted: date

Abstract In this paper, we present ORBWEB, a network substrate for Peer-to-Peer Grid Computing based on the open industrial-strength *eXtensible Messaging and Presence Protocol* (XMPP). We discuss, how XMPP can be leveraged to tackle domain-specific challenges, including high scalability, support for volatility, NAT/Firewall traversal, and protocol efficiency. Where XMPP fails to meet these requirements, we contribute pertinent extensions. In particular, we boost the scalability of XMPP by taking load of the XMPP servers through dynamically negotiated direct Peer-to-Peer communication channels between XMPP peers. We pave the way for scalable group membership management by substituting the existing XMPP Multi-User Chat protocol for one that does not suffer from limitations imposed by an »*everyone knows everyone*«visibility model and allows for selecting a group membership model that matches the requirements of a given application. As efficient multicasting is an essential prerequisite for many distributed algorithms, we adapt the well-known Bimodal Multicast protocol to work in a highly volatile Peer-to-Peer Grid Computing environment. Finally, we show how to improve the protocol efficiency of XMPP by leveraging a standardized binary encoding of the XML Information Set for XMPP packet transmission. To substantiate the applicability of our approach and the effectiveness of our extensions, we describe how some important higher-level services used in Peer-to-Peer Grid Computing can be implemented on top of ORBWEB and provide a detailed experimental analysis.

Keywords Grid Computing · Peer-to-Peer · Desktop Grid · Communication Middleware

Sven Schulz
University of Tuebingen
Tel.: +49-7071-2970472
Fax: +49-7071-295160
E-mail: schulzs@informatik.uni-tuebingen.de

Wolfgang Blochinger
University of Tuebingen
Tel.: +49-7071-2970469
Fax: +49-7071-295160
E-mail: blochinger@informatik.uni-tuebingen.de

Mathias Poths
University of Tuebingen
E-mail: poths@informatik.uni-tuebingen.de

1 Introduction

Peer-to-Peer (P2P) Grid systems harness underutilized resources of often geographically distributed desktop computers to tackle computationally challenging problems. Due to a remarkable cost-benefit ratio, P2P Desktop Grid Computing has become a promising alternative to classical Grids (e.g. based on the Globus toolkit) for certain kinds of applications. Both approaches to Grid Computing ultimately pursue the same goal: aggregation of resources beyond local administrative domains. Thus P2P Grid Computing can be regarded as a discipline of Grid Computing. However, there are also significant differences. Particularly, P2P Grid environments are significantly more heterogeneous and resource availability is much more dynamic. Delivering sustained computing power in such a dynamic and diverse environment is highly challenging, raising a plethora of research challenges. Our previous work focused on software architectures [1,2] for and applications [3,4] of P2P Desktop Grids.

In this work, which details and builds on the results of [5], we deal with the underlying communication infrastructure, called the *(network) substrate*. A substrate for P2P Grid Computing provides support for efficient P2P interaction and serves as a basic building block for higher-level protocols, services (e.g. resource discovery or information aggregation), and applications. In analogy to the development of Grid Computing, which experienced a phase of consolidation through standardization in the last decade, we believe that P2P Grid Computing has to pass through a similar process by adopting existing open standards to tap its full potential. By leveraging open standards the community will profit from a whole range of advantages including focus on interoperability from the beginning, improved robustness and durability leading to lower and manageable risk, and efficient use of existing resources allowing the community to concentrate on superordinate research challenges.

As a first step towards this goal, we propose to use the industrial strength *eXtensible Messaging and Presence Protocol* (XMPP) to build a generic substrate for P2P Grid Computing. While a couple of projects already made an ad hoc transition to XMPP (see Section 13), we strive to pave the way for a wider adoption of XMPP by systematic extension and optimization of the core protocols. Our key contributions are as follows: First, we specify functional and non-functional requirements defining a substrate concept suitable for P2P Grid Computing and demonstrate that the abstractions and the existing infrastructure of XMPP are basically well-suited to satisfy these requirements. Second, we contribute pertinent extensions and improvements amalgamated in our substrate called ORBWEB to further improve the applicability of XMPP for P2P Grid Computing. To underpin our approach in general and the effectiveness of our improvements in particular, we describe how higher level protocols can be implemented on top of ORBWEB and provide detailed benchmark results for all important aspects of the system. To be of use for the community, ORBWEB can be downloaded from `http://www.cohesion.de`.

The remainder of this paper is organized as follows: In Section 2, we describe an operational model suitable for Peer-to-Peer Grid applications beyond embarrassing parallelism. In Section 3, we identify functional and non-functional requirements for a substrate supporting this operational model. After giving an introduction to the XMPP core protocols and their extensions in Section 4, we describe in Section 5 how the elements of the XMPP protocol can be used and amended to realize a substrate satisfying the identified requirements. In Sections 6-9, we present the enhancements necessary to eliminate these shortcomings. Section 10 describes how some important higher-level services can be implemented on top of ORBWEB. Section 11 presents supportive tools to visualize peer topologies and to analyze server-side network traffic and physical network segmentation. In Section 12, we present a detailed evaluation of the impact of our extensions and improvements on XMPP protocol performance. Finally, Section 13 details related work and Section 14 concludes the paper by summarizing our contributions and identifying directions for future research.

2 Peer-to-Peer Grid Computing

In this section we first give a brief account of the field of P2P Grid Computing. Subsequently, we develop an operational model reflecting the specific characteristics of these systems.

2.1 Characteristics of Peer-to-Peer Grids

P2P Grids belong to the class of Desktop Grids [6]. They can be operated by virtually all institutions and can deliver considerable computational power at virtually no extra cost [7]. Traditional Desktop Grid systems rely on a client/server operational model. As a consequence, respective applications are most often based on trivial parallelism following the master/server or bag of tasks model. Prominent representatives of Desktop Grid platforms are BOINC [8] for volunteer computing and Entropia [9] for enterprise deployment scenarios.

The specific goal of the P2P Grid approach is to extend the applicability of Desktop Grid Computing towards non-trivial parallelism. The P2P principles enable complex interaction patterns among the participating hosts such that advanced parallel programming models can be realized. For example, parallel applications based on dynamic problem decomposition, like discrete optimization or constraint satisfaction, can benefit from the advanced capabilities of P2P Grids. Here new tasks are continuously generated at different locations and must be dynamically balanced over the available processors. Typical examples of P2P Grid systems are Personal Power Plant (P3) [10] and JNGI [11].

P2P Grids differ notably from other P2P based applications, like file sharing or instant messaging: In order to achieve high parallel efficiency, economical use of resources is of primary interest in P2P Grids. Resources are also limited due to constraints determined by the resource owners which are typically the users of the computers. Moreover, in P2P Grids, no user intervention can be assumed such that any kind of fault should be handled transparently.

Traditional Grids, realizing virtual organizations and P2P Grids ultimately pursue the same goal: aggregation of resources beyond local administrative domains. However, the two approaches face different requirements and constraints, like target communities (limited trust vs. no trust) or nature of resources (high-end vs. end-user) [12]. System architectures for building virtual organizations must specifically deal with interoperability issues, like standardization of protocols and interfaces.

In contrast, architectures for constructing and operating P2P Grids must reflect the high degree of resource volatility. Additionally, only little administrative overhead is acceptable, since typically no additional personnel is available for operating P2P Grid installations. As a consequence, lightweight, modular, and self-organizing system architectures become mandatory, since they reduce software and runtime complexity and can also adapt to the prevailing dynamism.

2.2 Operational Model

We assume an operational model of P2P Grids where several parallel applications and/or several instances of a parallel application execute concurrently within a distributed system comprised of a potentially large number of peers forming the whole Grid. In order to isolate the execution of different applications and application instances from each other, peers are organized into *groups*, which are the basic conceptual building blocks within our operational model. Basically, groups provide a scope for communication and a security context for the different applications. Moreover, groups are an important concept of parallel programming models (e.g. communication domains in MPI). Consequently, we need a hierarchical group model, enabling logical structuring of parallel computations within an application group.

As outlined above, the main benefit of P2P Grids compared to traditional Desktop Grids is that they allow for advanced parallel programming models supporting non-trivial parallelism. Due to the typically denser interaction patterns of respective applications we can assume that often only a medium number of peers (typically 100–1000 depending on the actual scalability properties of the parallel applications) can be beneficially employed for executing an application instance. Thus, application groups can be assumed to be of moderate size. Additionally, all peers of the system are members of a world group which provides the scope for operating, monitoring, and maintaining the whole system. Also, the peers of one or more application groups can be further arranged into organizational groups according to e.g. scientific institutions. Consequently, the size of these types of groups can be considerably larger than the size of typical application groups.

In the next section, we discuss the requirements for a network substrate suitable to support all aspects of our operational model of P2P Grids.

3 Requirements

In this section, we identify *functional* and *non-functional requirements* for a Peer-to-Peer Grid Computing substrate. We define the term *functional requirement* as the set of operations such a substrate must provide to support the P2P interaction model within a Desktop Grid environment. Non-functional requirements describe the qualities of the operations provided by the substrate. Our definition encompasses basic operations required to build higher level services, in particular structured as well as unstructured approaches to resource discovery. We derived and verified the requirements by integrating our implementation as the default substrate into our COHESION P2P Desktop Grid Computing platform [1] (see `http://www.cohesion.de`).

3.1 Functional Requirements

(F1) Group Abstraction. As discussed in the last section a group abstraction is a fundamental requirement for our operational model as well as for the general P2P model since it reflects knowledge and presence of other peers. In contrast to traditional parallel systems the set of nodes within the system is not static. Membership is in a constant state of flux, where nodes join and leave in an unpredictable, often uncorrelated manner. This phenomenon is called *volatility* [13] or *host churn*. In order to interact, a node must be able to track the changes in membership at least for a subset of all present nodes. Since the relations of a node to others may be manifold and may change over time, we also require groups to

be first class objects, i.e. a node may instantiate and destroy as many groups as required. Since groups are employed for different purposes in our operational model, we face different requirements for the group concept. On the one hand, we need group models which provide highest scalability, e.g. for driving the world group or for applications which exhibit appropriate scalability. On the other hand, we need group models which exhibit rather high efficiency for supporting application (sub-)groups.
(F2) Communication. To support interaction among peers, a substrate for P2P Grid Computing must at least provide point-to-point communication. Additionally, many distributed algorithms are based on a broadcast communication primitive, that allows for delivering a piece of information to all nodes in the system. Since we also require a group abstraction, broadcast becomes groupcast communication, where information is delivered to all members of a group.

3.2 Non-Functional Requirements

(N1) Performance. Achieving high parallel efficiency is a major goal in P2P Grid Computing. Thus, an appropriate substrate must provide low-latency and bandwidth economical communication. It should also deliver changes in group membership views promptly enabling efficient use of resources.
(N2) Scalability. Todays largest Desktop Grid Computing systems are comprised of up to hundreds of thousands of nodes [7]. This paramount scalability is possible since application support is limited to embarrassingly parallel applications within a client/server interaction model. However, if support for more non-trivial parallelism [1] is required, P2P interaction becomes mandatory. Scalability in such systems is achieved by distributing state over the participating peers. Unfortunately, this decentralization necessitates communication for synchronization and coordination among the peers. Hence, there is a trade-off between scalability and performance. In contrast to other P2P applications, where absolute performance is of less importance, the performance requirement cannot be neglected for P2P Desktop Grids, since it is crucial for achieving adequate parallel efficiency.
(N3) Connectivity. As opposed to traditional parallel systems connectivity in P2P Grids is typically limited due to NAT devices and restrictive firewalls. A common solution to this problem is *relaying*, where a mediator node, that is reachable by both parties, accepts messages from a node and forwards them to the respective other node. However, this introduces a bottleneck. Fortunately, more efficient solutions like NAT hole punching have been developed recently. A P2P Grid Computing substrate should be able to bridge network segmentations in an efficient way and thus provide universal connectivity.
(N4) Security. Since large-scale P2P Grids typically span more than a single administrative domain, a suitable substrate must undertake measures to keep sensitive data private and to protect the system state from malicious participants. This includes securing communication as well as restricting access to groups.

4 XMPP Overview

The *Extensible Messaging and Presence Protocol* (XMPP) is an open, XML-based protocol for real-time communication that has been formalized by the *Internet Engineering Task Force* (IETF). As XMPP is modular, it can be easily extended to adapt to use cases not covered by the core specifications published as RFC 3920 and RFC 3921. Extensions are managed by the *XMPP Software Foundation* [14] as publicly available *XMPP Extension Protocols* (XEP). Historically, XMPP has been used for instant messaging (Jabber IM) and presence information. Presence is the IM term for information whether a user is available or not. Due to its extensibility, the scope of XMPP has grown significantly since its invention. All kinds of applications based on real-time message exchange including media negotiation, whiteboarding, collaboration, content syndication, and generalized XML routing have been built on top of XMPP. Today, XMPP-based software is deployed on thousands of servers across the Internet.

Network architecture. The XMPP network is organized in a way that resembles the email network. Every user has a unique *Jabber ID* (JID). The JID consists of a user name and a DNS server name separated by an *at* sign, such as `foo@cohesion.de`. *XMPP entities*, i.e. clients and servers, communicate by exchanging XMPP messages called *stanzas*.

The XMPP network is decentralized and uses a simple message routing mechanism. If an XMPP server (`first.cohesion.de`) receives a message addressed to `bar@second.cohesion.de` from a locally connected user `foo`, the message is forwarded to the XMPP server at `second.cohesion.de`. For that purpose the server typically maintains a cache of inter-server connections. Servers that cooperate by routing messages to each other are called *federated* in XMPP jargon. XMPP users can interact seamlessly with users located on other networks (with a different protocol stack) through special components called *gateways*.

Connectivity. Today many clients are behind restrictive firewalls that allow outgoing traffic only on the HTTP port. XMPP defines an HTTP binding that can be used by these clients to connect to the server using a long-lived HTTP connection. The HTTP binding adheres to a push-model to deliver messages, i.e. they are delivered as soon as they are sent. In contrast to polling, where many polls return no new messages, this model is more efficient. As XMPP employs the client/server model, NAT traversal is no problem.

Security. XMPP provides several levels of security at the protocol level. Spoofing is impeded by forcing clients (for client-to-server connections) and servers (for server-to-server connections) to authenticate to their host server. Server dialback and whitelisting are additional security measures used to control which server-to-server connections are allowed. Since both, the connection establishment phase and the communication phase of the protocol are secured by SASL and TLS, client/server communication in XMPP is inherently secure. As stanzas are potentially routed over intermediary servers that may belong to third-parties, XMPP provides end-to-end signing and object encryption [15] to prevent rogue servers from spying on communications.

Multi User Chats. *Multi-User Chat* (MUC), defined in XEP-0045, extends the XMPP protocol to enable several clients to communicate in a many-to-many fashion. The scope of such a conversation is defined by *rooms*. A room is a set of $(JID, role)$ pairs maintained by the XMPP server. Depending on its role, each occupant, identified by its JID, has certain rights within the room (e.g. kick or invite users). The hosting XMPP server propagates changes in room membership to all occupants using *presence* stanzas. Thus, clients can keep their membership lists up-to-date. Occupants within a MUC room can send *message* stanzas addressed to the room. Such messages are delivered to all occupants by the XMPP server. One-to-one communication is supported by XMPP private chats. As described in the following section, we use MUC rooms to implement both the group abstraction and the groupcast communication primitive.

Implementations. With EJABBERD [16] and OPENFIRE [17] there are two industrial strength XMPP server implementations available, that are capable of serving large networks handling thousands of concurrent connections. EJABBERD provides superior performance and fault-tolerance through clustering support and would have been an excellent candidate to

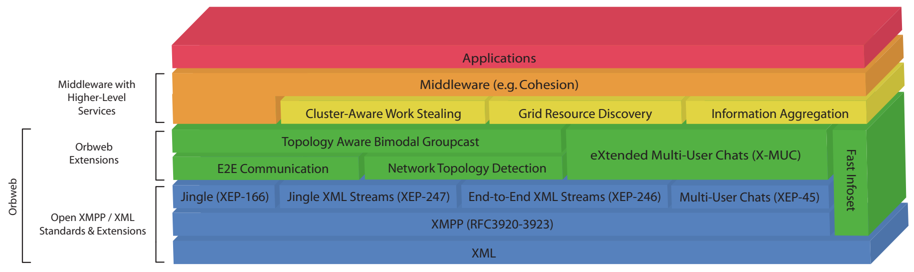

Applications
Middleware (e.g. Cohesion)
Cluster-Aware Work Stealing
Grid Resource Discovery
Information Aggregation
Topology Aware Bimodal Groupcast
E2E Communication
Network Topology Detection
eXtended Multi-User Chats (X-MUC)
Fast Infoset
Jingle (XEP-166)
Jingle XML Streams (XEP-247)
End-to-End XML Streams (XEP-246)
Multi-User Chats (XEP-45)
XMPP (RFC3920-3923)
XML
Middleware with Higher-Level Services
Orbweb Extensions
Open XMPP / XML Standards & Extensions
Orbweb

Fig. 1 ORBWEB's architecture with the exemplary higher-level services described in Section 10.

implement our substrate. Unfortunately, it is written in Erlang, for which to our knowledge there is no Fast Infoset (see Section 9) implementation available, which we use to improve the protocol efficiency of XMPP. OPENFIRE is written in Java™ for which a Fast Infoset implementation exists [18].

Although there are tens of client libraries available, some of them implementing a large subset of XEPs, most of them lack support for Jingle and its derivative XEPs (see Section 6). One of the most feature-rich implementations with Jingle support is SMACK [17]. Like OPENFIRE, SMACK is written in Java™.

We use OPENFIRE and SMACK as the basic XMPP stack on top of which we implement our optimizations.

5 Architecture

ORBWEB belongs to the class of unstructured hybrid or hierarchical P2P networks. In contrast to pure P2P, the hybrid approach is characterized by the fact that part of the network functionality is delegated to a comparatively small number of distinguished peers often called *superpeers*. In hybrid P2P networks built for data-centric applications (e.g. file sharing) superpeers are used as caches for resource indices of connected edge peers. By concentrating knowledge on more powerful peers, query processing times can be dramatically reduced as less communication with possibly slow edge peers is necessary. ORBWEB adopts this idea by delegating membership management to a central component. In analogy to faster query processing in the case of data-centric P2P applications, this allows for rapid membership updates that are essential for achieving good efficiency in P2P Grid Computing applications. In this section, we describe, how we leveraged where possible and amended where necessary the XMPP protocol stack and infrastructure to realize ORBWEB as a hybrid P2P substrate for P2P Grid Computing that satisfies the functional and non-functional requirements identified in Section 3.

Figure 1 shows the protocol and service stack of ORBWEB. We selected XMPP from the large number of possible communication technologies because the open XMPP standards, depicted in the lower layers of Figure 1, already cover our functional requirements: The Group abstraction (**F1**) can be mapped to XMPP MUCs. As an XMPP server can host any number of MUCs, peers are free to create as many groups as required. The unicast and groupcast communication primitives (**F2**) are also covered by the functionality provided by MUCs. While the former can be realized using private chats, the latter uses the fact that any occupant of a MUC room, can send a message to the room, i.e. to all room occupants.

Furthermore, XMPP already satisfies some of our non-functional requirements. It allows for universal connectivity (**N3**) through relaying by the XMPP server, which is almost always accessible as communication is client-initiated and an HTTP binding exists to tunnel restrictive server-side firewalls (XEP-124). With multiple security measures at the protocol level XMPP also fulfills our security requirements (**N4**).

However, as XMPP was not explicitly designed for Grid Computing, it is no surprise that there are some areas for improvements concerning our non-functional requirements: First, XMPP implements no P2P interaction model: Even those messages that could be exchanged directly between two clients are relayed by the XMPP server. Second, the MUC protocol maintains complete membership information at all nodes, resulting in maintenance costs that grow quadratically with group size. Third, groupcast messages are delivered by having the server send the message explicitly to each group member resulting in costs that grow linearly with group size. Fourth, XMPP is an XML protocol that is verbose and highly redundant. Thus XMPP over conventionally encoded XML unnecessarily wastes bandwidth and processing power. Taken together, these shortcomings results in the server becoming a performance bottleneck, when groups grow large and/or a large number of messages have to be relayed. Without further optimizations our substrate would fail to satisfy the non-functional requirements for performance (**N1**) and scalability (**N2**).

We addressed these issues by providing a set of extensions to the XMPP protocol and the OPENFIRE/SMACK XMPP software stack (see the middle layers of Figure 1): As described in Section 6, we modified the XMPP communication subsystem to create direct inter-client connections that can be used for unicast message delivery. Connections are created based on traffic pattern analysis in a way that respects limited client capabilities. Thanks to this modification ORBWEB takes considerable parts of the relay load off the XMPP server. In Section 7, we describe how the MUC protocol can be extended to support, among others, tree-based topologies creating partial membership views of configurable size resulting in maintenance costs that are logarithmic with respect to the size of the group. Section 8 delineates a probabilistic topology-aware decentralized groupcast implementation with server-side costs that are constant with respect to the size of the group. Finally, we describe how *Fast Infoset*, a binary encoding of XML, can be integrated into the XMPP server without sacrificing scalability in Section 9. As will be substantiated in Section 10 and Section 12, these optimizations significantly improve the applicability, the performance (**N1**), and the scalability (**N2**) of ORBWEB .

6 Efficient P2P Interaction

XMPP servers act as relays for exchanging messages between connected clients. By this means they guarantee for universal connectivity even for such hosts that are behind restrictive firewalls or NAT devices. However, the indirection over one or more servers limits scalability and performance unnecessarily, when hosts are able to communicate directly. Hence, a modification of the XMPP message delivery subsystem that enables true P2P communication promises to increase overall system-wide message throughput (by eliminating the performance bottlenecks induced by XMPP servers) and to decrease message latency (by reducing the number of necessary hops from three to one). We call this feature ORBWEB *End-to-End (E2E) communication*, as peers at the ends of such virtual XMPP connection interact directly.

To identify target peers for which the establishment of an E2E session is most beneficial, ORBWEB's E2E communication facility monitors outgoing XMPP traffic and tries to

Extension	Name	Description
XEP-0166	*Jingle*	Enables client-to-client sessions between XMPP entities. Jingle is a pure signaling protocol, i.e. it controls the connection negotiation process over the XMPP channel, while P2P interaction is accomplished *out-of-band* using custom communication technologies like the *Real-time Transport Protocol* (RTP), the *User Datagram Protocol* (UDP), or the *Interactive Connectivity Establishment* (ICE) protocol. Jingle is primarily targeted to support media exchange applications like voice or video chats. However, due to its modular design Jingle can be easily extended to support other session types and transport mechanisms.
XEP-0246	*End-to-End XML Streams*	Defines how two peers interact, i.e. which XMPP stanzas they exchange, after the session has been negotiated. This includes the exchange of stream headers, the use of TLS and SASL for establishing the security context and the closing of the XML stream.
XEP-0247	*Jingle XML Streams*	Defines a Jingle application type for establishing a direct XML stream between two XMPP entities over a reliable transport.

Table 1 XMPP extensions used by ORBWEB's E2E communication facility.

establish *E2E Sessions* with those partners with which many stanzas have been exchanged recently. To prevent from excessive resource consumption and to ensure scalability in the face of slow peers, that would be overloaded when forced to handle a large number of E2E sessions, the E2E facility enforces a session limit based on the capabilities of the peer. Besides selecting which peers are promising session partners, the system must implement the actual session establishment process. This is done by leveraging multiple existing XMPP extensions (see Table 1): we use *Jingle* (XEP-0166) and *Jingle XML Streams* (XEP-0247) for session negotiation and *End-to-End XML Streams* (XEP-0246) to establish an XMPP connection between peers according to the results of the negotiation process.

6.1 Implementation

ORBWEB's E2E communication facility (see Figure 2) attaches to all XMPP sessions a client establishes to sample outgoing traffic. This includes ordinary client-to-server (C2S) as well as E2E sessions. Based on the sampled data, i.e. the number of outgoing XMPP stanzas for a given target JID, a priority is computed for each target JID at regular intervals, called a *round* for the sake of brevity. To lessen the impact of past traffic patterns the priority value is aged after each round by multiplication with a factor $f_{age} \in]0,1[$. Based on these priorities the facility computes a list of JIDs, called the *nominal session list*, for which it is expected to be most beneficial to have an E2E session with. Subsequently, a series of session establishment and session termination tasks is computed by comparing the nominal and the actual session list. For each establishment attempt one of the following conditions hold:

1. The attempt succeeds and the new session is added to the actual session list.
2. The attempt fails because the session partner formally declines the initiation request or the session partner does not respond in a timely fashion or does respond with an error stanza, which happens for example when the session limit is exceeded. In both cases the session partner is greylisted according to a *greylisting strategy*, that determines for

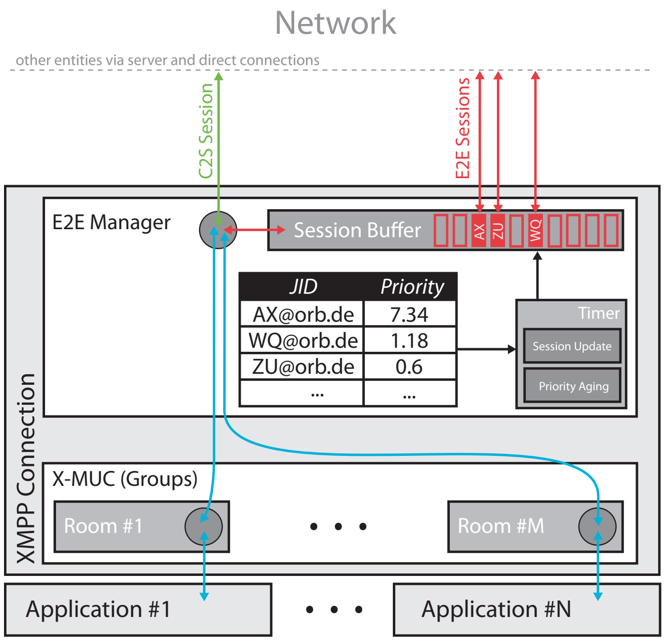


Fig. 2 ORBWEB's E2E communication facility creates E2E sessions to peers with which many messages have been exchanged in the past. These sessions are used to deliver XMPP stanzas directly. If no E2E session exists for a given target peer, the facility falls back to server relayed delivery over the single client-to-server (C2S) connection, which is available permanently.

how long a potential partner is excluded from session establishment after an attempt to establish an E2E session has failed. In our experiments a sigmoidal greylisting function turned out to be a good choice.

To prevent from session trashing, i.e. a condition in which the system is spending most of its time closing and establishing sessions caused by rapid priority order alteration near the priority limit where a session becomes qualified for E2E session establishment or is displaced by another one, the priority value of a recently closed session is reduced by multiplication with a *penalty factor* $f_{penalty} \ll 1$.

To be able to handle large numbers of connections ORBWEB's E2E communication facility implements the *reactor pattern* based on the APACHE MINA [19] high-performance protocol construction framework. In contrast to the traditional thread-per-connection model with limited scalability, a single thread is sufficient to serve all incoming and outgoing sessions.

6.2 XMPP Network Distance Service

Dependent on the application, a peer may exchange messages with a large number of other peers. Due to network segmentation and limited resources, it may be impossible to maintain E2E sessions to all of them. While the E2E facility ensures that E2E sessions are established for the most actively used communication paths, there is no application-level knowledge about the associated communication costs. Provided that the application allows for selecting with which peers to collaborate at what intensity, a peer should prefer interacting with peers over E2E sessions for efficiency reasons. To satisfy this requirement our E2E communication facility supports querying for the distance $d_{XMPP}(v)$ to another node v. Possible distances include $d_{XMPP}(v) = 0$ for the peer itself, $d_{XMPP}(v) = 1$ for peers with which an E2E session has been established, $d_{XMPP}(v) = 2$ for peers connected to the same, and finally $d_{XMPP}(v) = 3$ for peers connected to different XMPP servers. By reengineering network- and application-level protocols to be distance aware the utilization of the infrastructure can be optimized as less stanzas have to be relayed by the XMPP server. In Section 10.1 we describe how this feature of ORBWEB can be used to implement cluster-aware random stealing.

7 Scalable Group Membership Management

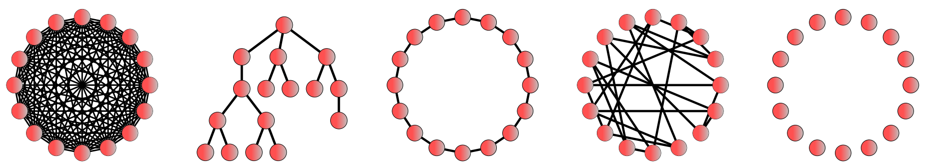
A *membership view* [20] is the subset of group members about whose status, i.e. available or unavailable, a node is informed. OPENFIRE MUC realizes complete membership views where each node is aware of the status of each other node. There are two reasons why this model is not the optimum choice for implementing the required group abstraction: First, the model does not scale to thousands of hosts neither client- nor server-side. The server has to transmit a quadratic number of messages to inform each group member of each other's membership status, i.e. join and leave operations are both of $O(|G|)$ time and message complexity when $|G|$ is the size of the group G. Additionally, the client keeps an account of all other group members, which leads to a memory usage linear in matters of the group size on each client. Second, the model is insensitive to application requirements: It is obvious that a load balancing algorithm based on random stealing (see Section 10.1) requires a fundamentally different membership model than a distributed hierarchical aggregation system requires (see Section 10.3). With a fixed membership model ORBWEB wouldn't qualify as a network substrate suitable for implementing a wide range of distributed applications.

To solve these issues, ORBWEB offers fine-grained control over the visibility model enforced within a group. By this means, an application can select the kind of topology – which is implicitly defined by the sum of local membership views – most suitable for the distributed algorithms in use. Note that ORBWEB nodes can still unicast messages to all other nodes within the group, even if they are not in their view, as long as they know their JID. We share the concept of partial membership lists with many DHT approaches (including Chord, Pastry and CAN), which use *neighbor sets* to route messages in a multihop fashion.

7.1 View Management

To be able to support a wide range of distributed algorithms and applications, we decouple group management (i.e. state management, security, etc.) from view management logic (i.e. which peer is aware of which other peers). Due to this separation of concerns, implementing new topologies can be done with minimal effort within the ORBWEB framework, either by starting from scratch or by composition of existing topologies.

To realize this concept, we developed a generic replacement for the MUC implementation of OPENFIRE called *eXtensible MUC* or *X-MUC*. X-MUC delegates view management to dedicated *view managers*. The architecture is based on the *composite* design pattern [21]: Each presence stanza received by the X-MUC component, indicating that a peer has joined or left the group, is forwarded to a *view manager* instance. The view manager translates the



(a) Complete (b) Red-Black Tree (c) Ring (d) Random (e) Blind

Fig. 3 Elementary view manager topologies for a 16-node group. An edge between two nodes means that they are part of each other's view.

Manager	Space Complexity		Message Complexity	Diameter
	Client	Server	Update	
Complete	$O(\vert G\vert)$	$O(\vert G\vert)$	$O(\vert G\vert)$	1
Blind	$O(1)$	$O(\vert G\vert)$	$O(1)$	∞
RBT	$O(v)$	$O(v\vert G\vert)$	$O(v)$	$O(\log(v\vert G\vert))$
Ring	$O(1)$	$O(\vert G\vert)$	$O(1)$	$\left\lfloor \frac{\vert G\vert}{2} \right\rfloor$
Random	$O(\log(\vert G\vert))$	$O(\vert G\vert\log(\vert G\vert))$	$O(\log(\vert G\vert))$	-

Table 2 View manager complexity characteristics for a group G of size $|G|$. Update complexities are the costs for handling a single join event.

incoming stanza into a set of outgoing stanzas based on its internal model of the group's topology. These presence stanzas are then delivered by the X-MUC component to the respective recipients updating their membership views accordingly.

A key strength of ORBWEB's view management is the ability to compose complex topologies from simpler ones. The composition mechanism is based on a special view manager that aggregates a set of subordinate view managers into a single one. To create such a *composite view manager* the implementor simply specifies what happens on a peer join or leave event: to which subordinate view manager(s) to add a joining peer with which parameters and from which to remove a leaving peer. The composite view manager translates presence updates from subordinate view managers into a single consistent view by interception or merging of individual presence stanzas. Note that view managers may attach attributes to presence stanzas, that can be used by peers to differentiate between contacts contributed by different subordinate view managers. For example our Bimodal Multicast implementation, described in Section 8, uses view attributes to partition local views into contacts used for unreliable groupcasting along a multicast tree and contacts used in the anti-entropy protocol phase. Furthermore, we make heavy use of view manager composition to implement the view managers described in Section 10.

7.2 Elementary View Managers

ORBWEB provides a set of elementary view manager implementations to satisfy the requirements for a broad range of use cases. Additionally, view manager composition can be used to create more complex topologies by combining two or more elementary view managers. Figure 3 shows sample topologies created by the elementary view managers. Their complexity characteristics are summarized in Table 2. ORBWEB's elementary view managers are:

Complete. This manager provides the same semantics as the standard XMPP MUC maintaining complete membership lists on all clients.

Red-Black Tree (RBT). This manager organizes group members in a tree and notifies nodes only about the group presence of its adjoining nodes in the tree. Hence, if a node is not the single member of a group, it will see a minimum of 1 (leafs) and a maximum of 3 (inner nodes) other members – we say its *view size* is 3. Each member can also configure its view size to other values, say s, which will cause the view manager to insert him $\min(1, \lfloor s/3 \rfloor)$ times with random keys into the tree. The member then has a maximum of s visible neighbors. To be able to perform tree updates in an efficient way, our implementation is based on a *red-black tree*. Updates in the tree (a joining or leaving member) can thus be computed in $O(s\log(|G|))$ time, where s is the maximum view size and $|G|$ is the size of group G. Configurable view size allows for implementing distributed algorithms that are aware of the capabilities of participating nodes, i.e. available resources, and exploit this knowledge to best utilize a heterogeneous resource set [22]. Another use case for configurable view size is to provide more detailed information about a groups composition to nodes that take special responsibilities within a group. A prominent distributed algorithm that makes use of such a distinguished node is the three-phase commit protocol [23].

The properties of red-black trees allow for providing an alternative implementation of our communication primitives where the server is not involved: As the red-black tree is a spanning tree, we can easily realize a groupcast by propagating messages along its edges. Additionally, as a red-black tree is a binary-search tree, we can use host-to-host message routing for unicasts known from DHTs. The choice of using a red-black tree assures, that two members are always connected by at most $4s\log(|G|)$ other members. Thus, the path length is $O(s\log(|G|))$.

Ring. This manager arranges the members of a group into a bidirectional ring. The order in which members appear in the ring can be customized by providing a custom comparator for node identifiers. A prominent example of ring-based distributed algorithms is termination detection for distributed computations [24]. Furthermore, we use the ring view manager to implement a topology for scalable reliable groupcast (see Section 8) and the Chord topology (see Section 10.2).

Random. Random networks are used in many distributed algorithms, especially in protocols using gossiping strategies. The Bimodal Multicast protocol described in Section 8 is a prime example of this class of protocols. The random view manager creates random networks where each peer v has a given outdegree $deg^{+}(v)$. While the default value of $deg^{+}(v)$ is $\log(|G|)$, each peer can configure the number of random contacts based on its capabilities, a concept similar to the view size used by the RBT view manager.

Blind. This manager does not send any information about other group members, which leads to optimal time and space usage, but minimum information. Note, that the group still enforces security restrictions and thus is appropriate to drive the root group realized in most communication frameworks with a hierarchical group model, e.g. the *world peergroup* in JXTA [25] or the *universal group* in COHESION [1].

8 Scalable Reliable Groupcast

Plain XMPP implements a simple groupcast scheme, where clients are arranged in a star-topology with the XMPP server at its center. Each groupcast message is sent to the XMPP server that forwards the message to all group members involving $O(|G|)$ server to client unicasts. This groupcast scheme is obviously not scalable and the XMPP server quickly becomes a performance bottleneck. Hence, ORBWEB implements a more sophisticated groupcast infrastructure.

Groupcast protocols with strong reliability properties, i.e. total order, atomicity, and virtual synchrony, are costly, include the possibility of unpredictable performance under stress or in the face of slow or stalled participants [26,27], and are of limited scalability [28]. Even with a very stable network of equally powerful participants, these protocols can hardly scale beyond several hundred participants [29]. In a P2P Grid scenario things are getting significantly worse: First, costly protocols deprive applications of bandwidth and processing power limiting the attainable overall efficiency of the distributed computation. Second, heterogeneity in performance among the hosts of a P2P Grid renders the existence of stale or (comparatively) slow hosts to be rather the rule than the exception. Third, P2P Grids are considerably more volatile than traditional parallel systems. Consequently, groupcast protocols with strong reliability properties are unsuitable for P2P Grid systems.

Another class of reliable groupcast protocols abandons the guarantee to deliver a message under all circumstances and at any price in favor of staying operational under worse conditions. Such protocols are called *best effort* protocols and differ from strongly reliable protocols in that participants detecting a failure restrict themselves to make only a reasonable effort to overcome it. The most carefully studied representatives of this class of protocols are the *Scalable Reliable Multicast* (SRM) protocol [30] and *Bimodal Multicast* [26]. The second has been developed because the first has shown to behave pathologically under certain conditions resulting in retransmission storms [31,32]. As the problematic behavior is triggered by transient elevated rates of message loss, SRM can be expected to be a poor choice for P2P Grids, where message losses caused by unexpected host departures or perturbations caused by slow or stalled hosts are frequent. Hence, we have selected Bimodal Multicast to replace the unscalable groupcast mechanism of XMPP in ORBWEB.

8.1 Bimodal Multicast

Bimodal Multicast – called *pbcast* by its inventors – is composed of two subprotocols: The first is an unreliable groupcast that makes a best-effort attempt to efficiently deliver a message to all group members. The second is a round-based two-phase anti-entropy protocol, that detects and corrects inconsistencies by continuous gossiping of locally available information. The first phase of the anti-entropy protocol detects message loss and if required triggers phase two, where such losses are corrected. Bimodal Multicast is simple and thus easy to implement. Figure 4 illustrates the execution of a single protocol round. After a period of unreliable groupcasts (participants P_1, P_2 and P_4 each send a groupcast message, where participant P_2 fails to receive M_0, P_4 lacks M_1, and P_3 misses M_2) the two-phase anti-entropy protocol is executed. Note, that the illustration simplifies the actual process, as the subprotocols are executed concurrently and participants advance independently. In the gossip phase each participant randomly selects another participant to which it sends a *digest* of its message buffer. All incoming payload messages are put into the message buffer and are removed after a configurable number of rounds. A digest contains the sequence numbers of all messages within the buffer. In the second phase of the anti-entropy protocol those participants that have received a digest perform a comparison with their own buffer entries. For each missing message they send a retransmission request, called a *solicitation*, to the sender of the digest. Upon receipt of a solicitation participants respond with the retransmission of the requested message. At the end of the second phase the message buffers of all participants

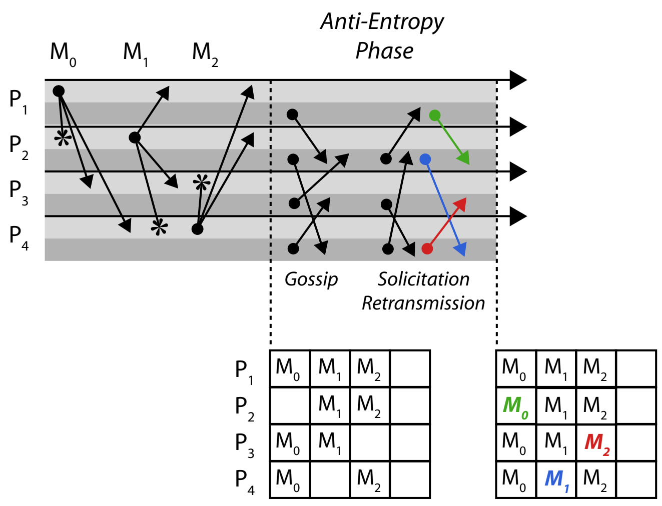


Fig. 4 Phases of the bimodal multicast.

have been populated with messages that were not reliably transmitted initially. The authors of the protocol propose a number of optimizations for this basic protocol. For a discussion of these optimizations see [26].

8.2 Topology-Aware Bimodal Multicast

The anti-entropy protocol of bimodal multicast randomly selects co-members as receivers for gossip messages. While this strategy is suitable for LAN settings with full connectivity, it is not well-suited for scenarios with network segmentations for two reasons: First, if segments are not bridged the gossiping is restricted to digest exchanges within segments. Consequently, the anti-entropy subprotocol won't be able to recover from message losses, where a message hasn't been received by any host within a given segment. Even if segments are bridged by NAT traversal techniques making recovery possible on any kind of loss, unicast costs are no longer uniform as cross-segment communication is typically of less bandwidth and higher latency than intra-segment communication. Furthermore, shared intermediate network devices experience heavy load in the face of massive inter-segment communication. This includes NATs, firewalls, and the XMPP server that is used to connect mutually unreachable hosts.

To remedy this limitation, we extend the bimodal multicast algorithm to take the segmentation of the underlying network into account. Therefore each participant is provided with a set of contacts for execution of the anti-entropy protocol that is composed of co-members selected randomly from the set of participants located within the same network segment only. This requires to augment the protocol with the ability to automatically detect network segments. We refer to this extended protocol as the *topology-aware bimodal multicast* or *ta-pbcast*.

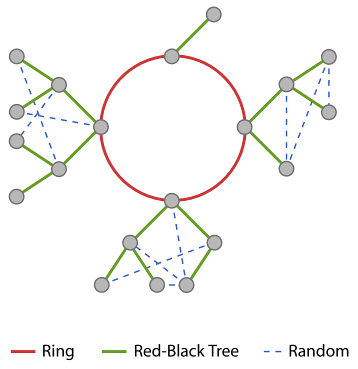


Fig. 5 Composite topology maintained by the *ta-pbcast* view manager.

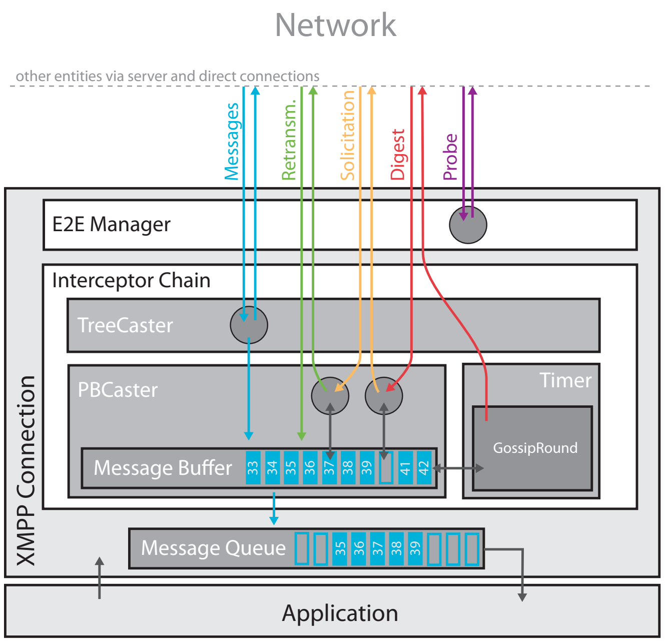


Fig. 6 Components of the client-side *ta-pbcast* logic.

8.3 Implementation

The *ta-pbcast* implementation of ORBWEB is based on a custom view manager composed of a ring view manager and $|C|$ superimposed red-black tree and random view managers, where C is the set of network segments spanned by the group members. While the resulting overall topology is illustrated in Figure 5, Figure 6 shows the client-side groupcast logic. The RBT view managers are used by the `TreeCaster` (cf. Figure 6) for the first phase of the Bimodal Multicast protocol as spanning trees for unreliable groupcasting. Instead of propagating an incoming groupcast message to the root of each red-black tree, the server selects a configurable number of injection points, i.e., peers within the same segment, based on their stability. The rationale for this scheme, called *stability-aware multi-injection*, is to minimize the probability that a message is not received by any peer within a network segment. This probability is minimized for two reasons: First, by selecting the most stable peers, unexpected departures of the receiving peers are less likely. Second, by using k injection points, even $k-1$ departures are not critical. Note, that we still need the anti-entropy phase of the bimodal multicast protocol as message losses may occur when a peer arrival or departure triggers a series of rotations within one of the red-black trees while a message is currently delivered along its edges. To avoid propagation of duplicate messages – either caused by rotations or due to multi-injection – the server injects a unique sequence number into incoming groupcast messages.

The Random view manager is used by the `PBCaster` (cf. Figure 6) in the anti-entropy phase of the Bimodal Multicast protocol. As all tree and random edges exist only between peers within the same network segment, no messages have to be relayed by the XMPP server. The ring view manager connects the root peers of all network segments. A groupcast operation first sends the message to the XMPP server, which sends the message to all peers that are part of the ring, which in turn use the red-black tree to deliver the message within their segment. Hence, a groupcast operation has a server-side complexity of $O(|C|)$, which results in costs that are in general several orders of magnitude smaller than the costs of the standard XMPP groupcast mechanism.

The above description covers a setting, where group membership is static and network segments are known a priori. However, most often group membership is dynamic and there is no or limited knowledge concerning network segments. Note, that even if the latter weren't true, the costs of (re-)configuring the substrate to respect network segments manually are prohibitive in a P2P Grid environment, where no or limited administrative manpower is available. Hence, ORBWEB implements an adaptive algorithm for topology detection that allows for self-management and avoids manual intervention: On arrival, a peer is submitted to the ring view manager that is served with groupcast messages by the XMPP server directly and thus becomes the root of a new red-black tree. A server-side component periodically issues *probe* requests containing a target peer to group members selected randomly from the set of ring members. On receipt of such a probe request, the E2E Manager at the recipient tries to establish an E2E session with the target peer and sends the result back to the server. On success – i.e., the peers are within the same network segment – the server merges the red-black trees mounted at the source and the target peer into a single tree. Note, that this scheme works only if two assumptions are valid: segments are (comparatively) static and connectivity is transitive, i.e. if peer A can establish a E2E session with peer B and peer B can establish a session with peer C, then peer A can establish a session with peer C.

9 Efficient XML Processing

An XMPP server spends most and XMPP clients a significant share of their time in XMPP stanza processing. Thus, an attempt to increase the overall performance of the communication subsystem should primarily address optimizing XML processing. In this section, we describe how we optimized the XML processing stack of OPENFIRE/SMACK by incorporating a binary XML encoding called *Fast Infoset* (FI) [18] to yield substantial latency and throughput improvements.

9.1 Fast Infoset

A major drawback of XML is that it is quite verbose. Since document size affects all stages in the XML processing chain (i.e. serialization, transmission, and parsing), techniques to reduce document size promise to increase processing performance. However, there is a trade-off between document size and pre-/postprocessing effort. Simple stream compression methods (e.g. GZIP) significantly reduce document size, but at the same time cause considerable pre-/postprocessing overhead. *Fast Infoset* (FI) overcomes this limitation by interweaving serialization and compression or decompression and parsing, respectively.

FI specifies a binary encoding format for the XML Information Set. It aims to provide more efficient serialization and parsing than the character-based standard XML format. FI is used to optimize both document size ($\approx 50\%$ on average compared to standard XML 1.0 serialization using Xerces 2.7.1 [18]) and processing performance ($\approx 25\%$ faster serialization and between 5 and 8 times faster parsing compared to Xerces 2.7.1 SAX) and thus is more advanced than simple stream compression based on GZIP used in contemporary XMPP servers. These improvements are achieved through exploiting redundancy by avoiding end-tags, applying string indexing and huffman encoding, aligning information for faster access and by directly embedding binary data into the stream, bypassing the usually necessary conversion to Base-64 representation.

9.2 Implementation

OPENFIRE and ORBWEB's E2E facility both leverage the *Reactor* design pattern [33] to achieve high scalability. Pending I/O-operations are handled sequentially by a single dispatcher thread (per CPU core). This processing model is very efficient, since fewer threads means less resource consumption and fewer context switches. However, since XMPP stanzas are delivered as elements embedded in a single large XML stream, this non-contiguous I/O processing style is problematic, as to our best knowledge, there are no non-blocking Java™ XML parsers available. A non-blocking XML parser would return immediately when no more data is available from the underlying input stream, leaving the parser in a continuable state and allowing the dispatcher thread to continue with processing pending operations from other connections.

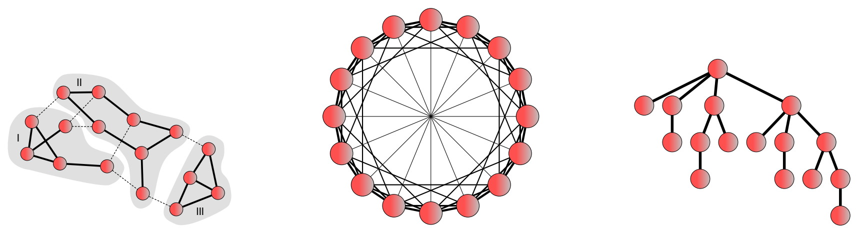
To solve this problem, we dissect the incoming byte stream containing the XMPP stanzas (see Figure 7). For that purpose we modified the XML serializers to prepend a header to each stanza specifying the length of the packet. Receiver-side logic reads the header and waits until the specified number of bytes have been received. As soon as the whole sequence of bytes has been received the whole stanza is forwarded at once to the XML parser, which immediately returns after the end tag concluding the XMPP stanza has been read.

10 Higher-level Services

In this section, we exemplify in detail how higher-level services can be implemented on top of the functionality provided by ORBWEB using load balancing, resource discovery and hierarchical information aggregation as examples.

```
           <stream:stream to="theta"
                          xmlns="jabber:client"
  TCP                     xmlns:stream="http://etherx.jabber.org/streams"
 Packet                   version="1.0">

   I       0x0000    00 00 00 61                                        ...a

   II      0x0000    E0 00 00 01 00 78 CD 0C-6A 61 62 62 65 72 3A 63    à....xÍ.jabber:c
           0x0010    6C 69 65 6E 74 CF 05 73-74 72 65 61 6D 1F 68 74    lientÏ.stream.ht
           0x0020    74 70 3A 2F 2F 65 74 68-65 72 78 2E 6A 61 62 62    tp://etherx.jabb
           0x0030    65 72 2E 6F 72 67 2F 73-74 72 65 61 6D 73 F0 3F    er.org/streamsð?
           0x0040    81 82 05 73 74 72 65 61-6D 78 01 74 6F 44 74 68    ?,.streamx.toDth
           0x0050    65 74 61 78 06 76 65 72-73 69 6F 6E 42 31 2E 30    etax.versionB1.0
           0x0060    F0                                                 ð

           <iq id="F9P2Vp-0" to="cohesion" type="get">
               <query xmlns="jabber:iq:register"></query>
           </iq>

  III      0x0000    00 00 00 47                                        ...G

   IV      0x0000    E1 00 74 FF 7C 01 69 71-78 01 69 64 46 39 50 32    á.tÿ|.iqx.idF9P2
           0x0010    56 70 2D 30 00 47 63 6F-68 65 73 69 6F 6E 78 03    Vp-0.Gcohesionx.
           0x0020    74 79 70 65 42 67 65 74-F0 38 CD 11 6A 61 62 62    typeBgetð8Í.jabb
           0x0030    65 72 3A 69 71 3A 72 65-67 69 73 74 65 72 F0 3C    er:iq:registerð<
           0x0040    04 71 75 65 72 79 FF                               .queryÿ
```

Fig. 7 FI encoded XMPP stream with headers (TCP packets I and III) specifying the length of the following XMPP stanza.



(a) Distance-Aware Load Balancing

(b) Chord Ring

(c) Physical aggregation tree

Fig. 8 View managers used to implement higher-level services.

10.1 Load Balancing

Load balancing is used to dynamically disperse tasks of a computation across the processors of a parallel or distributed system in order to obtain the highest possible execution speed. While static load balancing assigns tasks to processors before execution, dynamic load balancing is active in parallel to the execution of the tasks. Due to volatility and heterogeneity static load balancing is inapplicable for P2P Grids for two reasons: First, it would be very difficult to estimate the execution time of a task accurately without actually executing the task. Second, communication delays are inhomogeneous in WAN setups and in general unpredictable. Thus, dynamic load balancing is used in P2P Grid Computing. As a central master processor holding the whole collection of tasks to be executed on slave processors won't scale for applications when tasks are not independent, P2P Grids utilize a fully distributed work pool with decentralized dynamic load balancing. In this execution model tasks are passed between arbitrary processors to balance their load. For a thorough discussion of the topic see [1].

A simple load balancing algorithm is called *random stealing* [34]. As a receiver-initiated method, processes request tasks from other processes when they have few or no tasks left in their local work queues. As the name indicates the balancing partner is selected at random.

Implementing flat random stealing in ORBWEB is straightforward by leveraging the standard random view manager. However, this simple approach does not take into account that intersegment traffic is expensive and thus unnecessarily lays stress on the network infrastructure. Fortunately, there are two possible approaches to solve this problem with the services provided by ORBWEB: First, we can use the *ta-pbcast* view manager that already provides the described functionality as it deploys a random view manager within each network segment. Alternatively, we can deploy a single topology agnostic random view manager within the whole group and use the distance service described in 6.2 to gain information about the costs of interacting with a given remote peer. This approach has the advantage that we can mix inter- and intrasegment requests with a bias on intrasegment requests resulting in an algorithm referred to as *cluster-aware random stealing* [35]. Figure 8a depicts the respective topology. Thick edges represent intrasegment contacts that are used more frequently as targets for work stealing requests than intersegment contacts visualized as dashed edges.

10.2 Resource Discovery

P2P Grid resource discovery systems (GRDS) allow for placing, searching, locating and retrieving of information within P2P Grids. There are two types of such systems, based on unstructured and structured approaches. While unstructured GRDS' do not impose structure on the interconnection of the participating peers, structured GRDS' apply hash functions to peers and resources to implement placement and lookup.

Unstructured GRDS' use flooding (broadcasting with a limited scope) to discover resources and support complex queries but in general do not allow for any guarantee on the quality and completeness of the search results. Several techniques [36,37,38] have been proposed to enhance two intrinsic drawbacks of the flooding approach: the potentially tremendous amount of messages, and the possibility that an existing resource may not be located at all.

Structured GRDS' use hashing and logical topologies (such as a ring or a hypercube) to route both resources (placement) and queries (lookup) to a peer with the closest hash key. A query message is incrementally forwarded towards the target peer in a small and limited number of hops. As in enhanced unstructured approaches, the success of a lookup is guaranteed, if the desired key exists in the system. As compared to the flooding technique, structured approaches need expensive maintenance on membership updates but are much more efficient with respect to the number of messages transmitted per query. Recent research has shown that structured GRDS' can be enhanced to support keyword and range queries [39,40].

Realizing a GRDS on top of ORBWEB means implementing a view manager that recreates the same topology as the original algorithm does. As unstructured GRDS do not require a certain topology by definition, they can be used on top of ORBWEB without modification by using a topology aware random view manager. Different structured GRDS' create different topologies. Discussing how to implement all of them would go beyond the scope of this work. For the sake of brevity, we focus on a single structured GRDS approach, namely Chord [41].

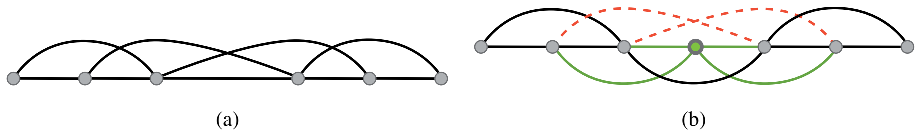


Fig. 9 Segment of a chord ring with first level fingers before (a) and after (b) a peer joins.

Chord arranges peers in a ring structure based on a unique identifier created by applying a hash function on local information that is unique for a given peer. Keys, i.e. information to be placed into the system, are mapped to peers using *consistent hashing* which ensures that only a small number of key reassignments are necessary in the face of peer arrivals or departures: Identifiers are ordered in an identifier circle modulo 2^m, where key k is assigned to the first peer whose identifier is equal to or follows k in the identifier space. In addition to the predecessor and successor within the ring, each peer maintains a set of links called *fingers*, where the i^{th} finger references the 2^{i-1}-th successor in the ring. Fingers ensure lookup efficiency as they allow to implement a distributed binary search. Key lookup happens iteratively: a peer q receiving a lookup request from peer p for key k searches its finger table for the peer r whose identifier most immediately precedes k and sends the result back to p. Peer p then sends a lookup request to peer r. Hence, p iteratively learns about peers with identifiers closer and closer to k.

To implement Chord on top of ORBWEB, we encode Chord identifiers into the nickname of an XMPP room member identifier and use a composite view manager consisting of a ring view manager that sorts peers according to this identifier and a custom view manager that is responsible for recreating the fingers. While the ring is updated on every join/leave event and hence guarantees routability within the Chord network, applying the same strategy of eagerly repairing the topology after changes in membership would be very inefficient. As illustrated in Figure 9 the arrival of a peer results in updating a large number of fingers (the same is true for departures). Note that the illustration shows updates for fingers of length 2 only and that the number of necessary updates increases with finger length. To avoid these massive reconfigurations, we restrict immediate updates to the addition of the fingers for the newly arrived peer. All other updates are performed by a background thread that periodically selects a small set of peers at random and updates their fingers. Hence, we trade off routing efficiency against topology accuracy and decouple topology maintenance costs from the degree of volatility. The topology created by the ORBWEB Chord view manager is depicted in Figure 8b for a 16-node group. The topology includes the basic ring and fingers up to the 4^{th} order.

A serious performance inhibitor of this simple implementation is its topology agnosticism: As peer identifiers are generated at random, neighborhood in the Chord ring is random, too. Hence, lookup request processing potentially involves a large number of message exchanges across segment borders each placing avoidable load on the XMPP infrastructure. To eliminate this waste of resources, we use the network segment detection algorithm described in Section 8. Whenever a peer is detected to be within a network segment, we assign a new identifier to it. This identifier is generated at random but constrained such that the peer is adjacent to one of the peers already assigned to that segment (see Figure 10 where peer x has been assigned a new identifier x^* after the peer has been detected to be in network segment (II)).

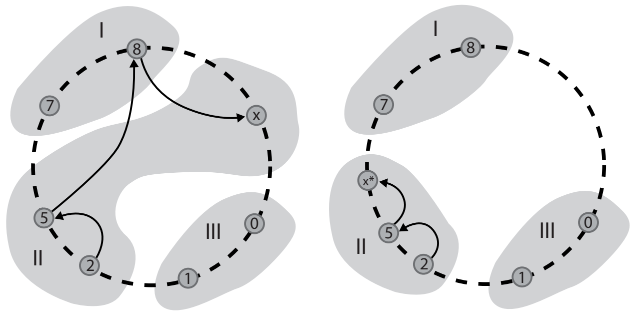


Fig. 10 Chord embeddings without (a) and with (b) topology awareness.

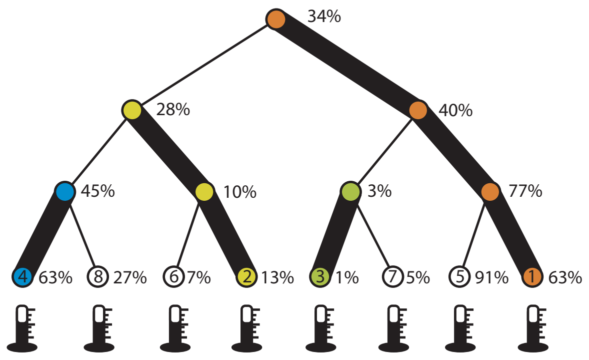


Fig. 11 Aggregation of CPU utilization along a logical aggregation tree with physical (leaf) and virtual (inner) nodes.

Although server-side topology management for Chord does not scale as well as the fully distributed protocol, complete knowledge of the membership list results in superior convergence of the Chord network as substantiated by the results presented in Section 13. Fast topology convergence is of particular importance for high-performance applications executing on highly volatile networks typical for P2P Grid Computing systems.

10.3 Information Aggregation

Information aggregation is the process of summarizing information across the nodes of a distributed system. It is considered a basic building block for distributed systems on top of which many fundamental distributed paradigms and algorithms can be realized, including leader election, service placement and error recovery.

For large-scale systems information aggregation must be performed in a hierarchical manner, as exposing all information to all nodes would quickly overwhelm even the most powerful nodes. Hierarchical approaches expose information with different levels of detail by progressively summarizing information over space and time. For these purposes hierarchical aggregation systems [22,42] maintain a logical aggregation tree where all leaf nodes are physical nodes and all inner nodes are virtual nodes simulated by selected physical nodes (see Figure 11). While the former actually gather information and provide it to the upper levels, the latter perform the aggregation reproviding its results to physical nodes.

Implementing such a logical aggregation tree on the server is straightforward, as the client-side tree maintenance method described in [22] based on a total order on the node identifiers can be easily moved to the server that knows all client identifiers anyway. Figure 8c shows the target topology created by the view manager used to implement the virtual tree. Even a topology-aware version of such a view manager can be constructed with little effort by modifying the order imposed on the set of node identifiers to account for nodes being part of the same network segment.

11 Tooling

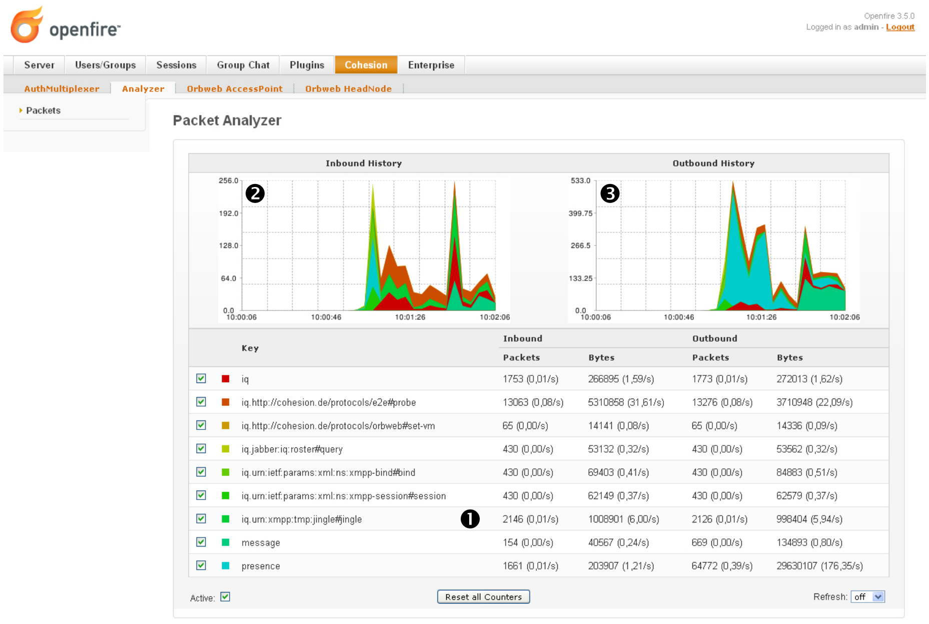
Understanding what happens within large distributed systems is complicated by the large number of interacting nodes, lack of centralized access to their execution context, and ubiquitous concurrency. For this reason tooling is of particular importance on all levels of P2P Grid Computing systems. This is especially true for the network substrate that forms the basis for the upper layers of the system. ORBWEB meets this demand by providing a rich set of supportive tools including (server-side) traffic analysis and online visualization of view topology and network segmentation. We describe these tools subsequently. They can also be downloaded from our website `http://www.cohesion.de`.

11.1 Traffic Analysis

XMPP and its extensions employ a large number of different stanza types. When developing complex protocols, like *ta-pbcast* (see Section 8), network traffic data provided by domain independent analysis tools is insufficient to gain a precise understanding of the actual packet flow produced by the protocol implementation. Hence, ORBWEB provides a packet analysis tool that is implemented as a plugin for and thus tightly integrates with the OPENFIRE XMPP server. Figure 12 shows a screenshot of the tool: It provides counters for incoming, outgoing, and the overall number and the transmission rate for presence, IQ (including namespace and action), and message stanzas (❶). To get a quick overview of the share a stanza type contributes to the overall traffic, a stacked chart visualization is provided (❷, ❸).

11.2 Component Visualization

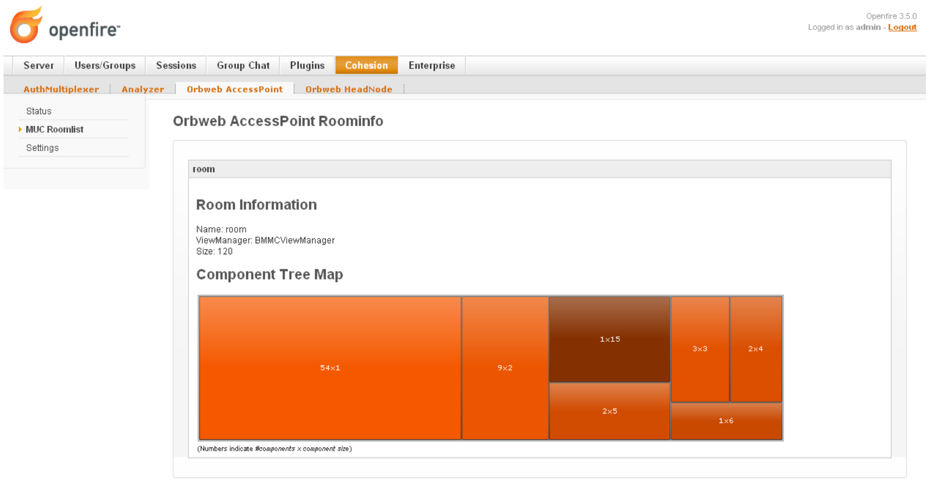
As described in Section 8, the *ta-pbcast* view manager automatically identifies network segments among the set of participating peers. As the message complexity directly depends on the number of network segments, knowledge about the network topology is of prime importance for debugging and evaluation purposes. Hence, ORBWEB provides a tree map visualization of the components currently identified by the *ta-pbcast* view manager (see Figure 13). In conjunction with the traffic analyzer the component visualization can provide valuable insight into the reasons of pathological traffic patterns.



Packet Analyzer

		Key	Inbound		Outbound	
			Packets	Bytes	Packets	Bytes
✓	🟥	iq	1753 (0,01/s)	266895 (1,59/s)	1773 (0,01/s)	272013 (1,62/s)
✓	🟧	iq.http://cohesion.de/protocols/e2e#probe	13063 (0,08/s)	5310858 (31,61/s)	13276 (0,08/s)	3710948 (22,09/s)
✓	🟨	iq.http://cohesion.de/protocols/orbweb#set-vm	65 (0,00/s)	14141 (0,08/s)	65 (0,00/s)	14336 (0,09/s)
✓	🟩	iq.jabber:iq:roster#query	430 (0,00/s)	53132 (0,32/s)	430 (0,00/s)	53562 (0,32/s)
✓	🟩	iq.urn:ietf:params:xml:ns:xmpp-bind#bind	430 (0,00/s)	69403 (0,41/s)	430 (0,00/s)	84883 (0,51/s)
✓	🟩	iq.urn:ietf:params:xml:ns:xmpp-session#session	430 (0,00/s)	62149 (0,37/s)	430 (0,00/s)	62579 (0,37/s)
✓	🟩	iq.urn:xmpp:tmp:jingle#jingle ❶	2146 (0,01/s)	1008901 (6,00/s)	2126 (0,01/s)	998404 (5,94/s)
✓	🟩	message	154 (0,00/s)	40567 (0,24/s)	669 (0,00/s)	134893 (0,80/s)
✓	🟦	presence	1661 (0,01/s)	203907 (1,21/s)	64772 (0,39/s)	29630107 (176,35/s)

Fig. 12 The ORBWEB traffic analyzer records and visualizes the overall number and the throughput rates for incoming and outgoing XMPP stanzas broken down into stanza types. The visualization is based on the Google *Chart API* and embedded into the OPENFIRE administration console.



Orbweb AccessPoint Roominfo

room

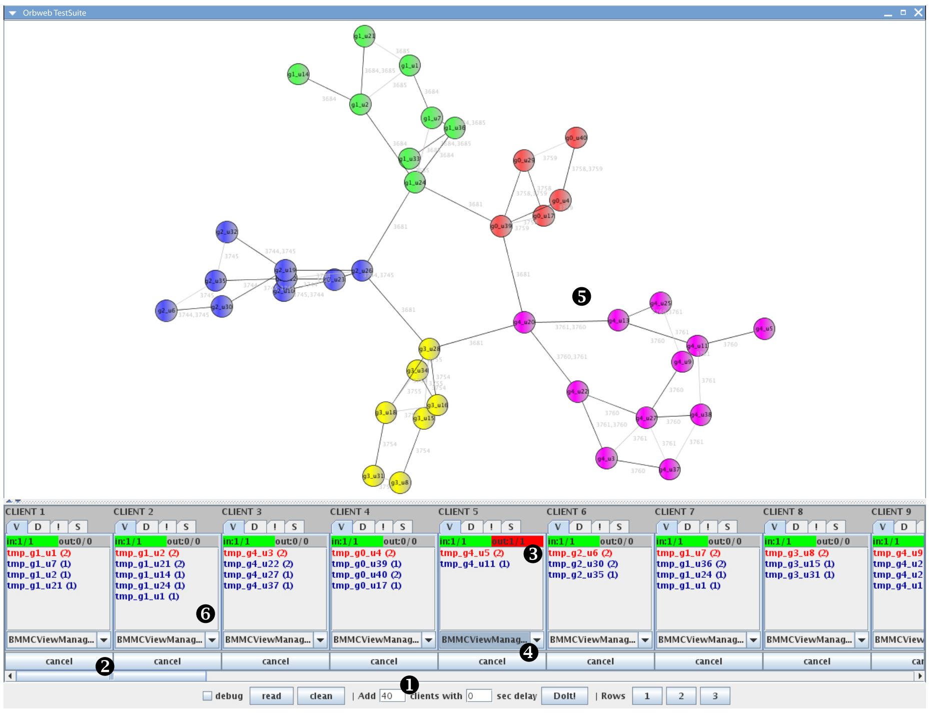
Room Information

Name: room
ViewManager: BMMCViewManager
Size: 120

Component Tree Map

(Numbers indicate #components x component size)

Fig. 13 A snapshot of the components within a 120-node group managed by *ta-pbcast* view manager visualized as a tree map just after creation. As indicated by the labels, there are 54 components of size 1, 9 of size 2, 3 of size 3, 2 of size 4 and 5, and 1 of size 6 and 15. The visualization is based on Adobe Flex™ and embedded into the OPENFIRE administration console.



```
CLIENT 1          CLIENT 2          CLIENT 3          CLIENT 4          CLIENT 5          CLIENT 6          CLIENT 7          CLIENT 8          CLIENT 9
in:1/1 out:0/0    in:1/1 out:0/0    in:1/1 out:0/0    in:1/1 out:0/0    in:1/1 out:1/1    in:1/1 out:0/0    in:1/1 out:0/0    in:1/1 out:0/0    in:1/1
tmp_g1_u1 (2)     tmp_g1_u2 (2)     tmp_g4_u3 (2)     tmp_g0_u4 (2)     tmp_g4_u5 (2)     tmp_g2_u6 (2)     tmp_g1_u7 (2)     tmp_g3_u8 (2)     tmp_g4_u9
tmp_g1_u7 (1)     tmp_g1_u21 (2)    tmp_g4_u22 (2)    tmp_g0_u39 (1)    tmp_g4_u11 (1)    tmp_g2_u30 (2)    tmp_g1_u36 (2)    tmp_g3_u15 (1)    tmp_g4_u2
tmp_g1_u2 (1)     tmp_g1_u14 (1)    tmp_g4_u27 (1)    tmp_g0_u40 (2)                      tmp_g2_u35 (1)    tmp_g1_u24 (1)    tmp_g3_u31 (1)    tmp_g4_u2
tmp_g1_u21 (1)    tmp_g1_u24 (1)    tmp_g4_u37 (1)    tmp_g0_u17 (1)                                        tmp_g1_u1 (1)                       tmp_g4_u1
                  tmp_g1_u1 (1)
```

Fig. 14 ORBWEB test client running 40 logical ORBWEB peers within a group with a *ta-pbcast* view manager (see Section 8).

11.3 Topology Visualization

ORBWEB's server-side group management approach allows for easy debugging and testing of group membership models by running a configurable number of test peers. Figure 14 presents our topology visualization tool showing a group of 40 peers managed by the *ta-pbcast* view manager (see Section 8). The tool allows for adding (❶) and removing peers (❷), unicasting and groupcasting of messages (❸), the selection of view managers (❹), and the visualization of the group's topology (❺). The latter leverages the sophisticated graph visualization library *yfiles* [43]. The view is dynamically updated on every change in the topology, i.e. addition and removal of peers and links. Note, that a link does not model a connection but a contact, i.e. that one peer has another peer in its view (❻).

12 Performance Evaluation

This section presents an analysis of the results obtained from running performance tests for both the original SMACK / OPENFIRE XMPP stack and ORBWEB.

12.1 Evaluation Method

The testbed used for our performance evaluation consists of 41 hosts located in three different Fast Ethernet Local Area Networks connected by a campus network as depicted in

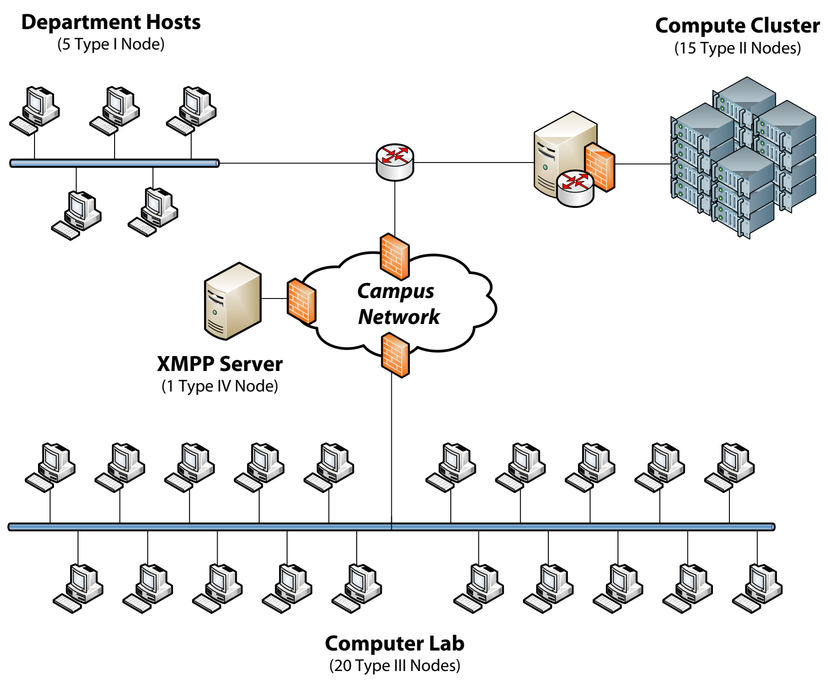


Fig. 15 Topology of the testbed used for performance tests.

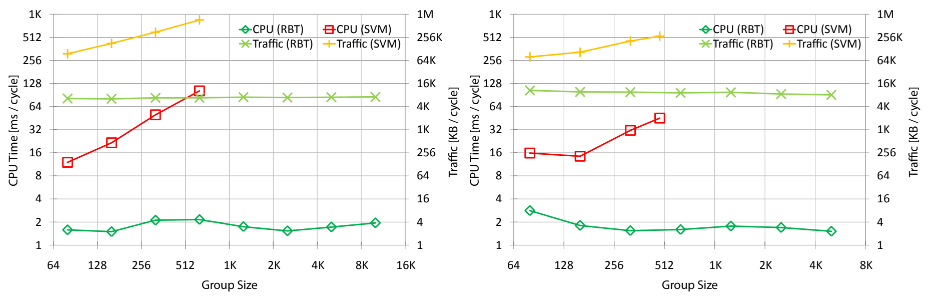
Type	Hardware		Software	
	CPU	Memory	OS	Kernel
I	AMD®Athlon™64 X2 4600+ 2 Cores 512KB Cache / Core	3GB	Linux	2.6.22-14-generic
II	Intel®Xeon™2.67GHz 2 Processors 512KB Cache / Processor	2GB	Linux	2.6.22.9
III	Intel®Pentium™D 3.40GHz 2 Cores 2048KB Cache / Core	2GB	Linux	2.6.23-gentoo-r8
IV	Intel®Core™2 Q6600 2.40GHz 4 Cores 2048KB Cache / Core	8GB	Linux	2.6.22-14-server

Table 3 Hard- and software setup of the testbed hosts.

Figure 15. 40 hosts are used to host XMPP test clients based on SMACK v3.0.0 and a single machine hosts our extended XMPP server based on OPENFIRE v3.5.0. The hard- and software setup of the hosts is summarized in Table 3.

To be able to push the XMPP server to its limits, we have each physical host run up to 256 clients in parallel leading to a maximum overall number of $\approx$ 10K clients. For test scenarios involving E2E sessions, we limit the number of collocated clients to 24 to avoid the risk of client-side overload.

We use *The Grinder* [44], an open source distributed load testing framework written in Java™, for test deployment and orchestration. To minimize context switching overhead collocated clients are driven by threads within a single process controlled by the load injec-



(a) Single volatile scenario (b) All volatile scenario

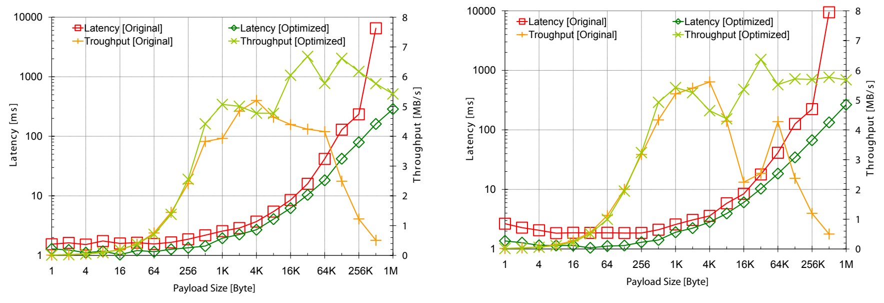
Fig. 16 Server-side resource usage per join/leave-cycle

tion agent of *The Grinder*. Furthermore, the online collection of statistics and results was replaced by a post-mortem approach to eliminate any interference with the actual test execution. A warm-up phase preceded each test run to eliminate the impact of *Just-In-Time* (JIT) compilation.

12.2 Membership Management

In this section, we report on the performance of the membership management operations, i.e. join and leave. The first test is carried out by having one of the group members join and leave periodically (once per second). To assess the suitability of our X-MUC (see Section 7) for driving large-scale groups, we compare the server-side resource consumption, i.e. network traffic and CPU usage, of both the original view-complete (SVM) and our RBT-based view manager (RBT) implementation with partial membership lists and view size 3. In order to obtain values related to a single join/leave-cycle, we calculate the quotient of overall resource consumption within the test period and the number of join/leave cycles actually performed. Figure 16a shows the results of our experiments. The theoretical linear message complexity of the original view-complete MUC room implementation (SVM) is perfectly confirmed by our measurements. Both CPU time and network traffic increase linearly reaching a maximum of 102 ms/cycle and 743 KB/cycle respectively for 640 nodes. Tests with node counts beyond 640 nodes failed due to excessive memory usage. From these results, we conclude that view completeness becomes infeasible for groups growing larger than $\approx$ 512 nodes, where roughly 10 joins can be processed each second. The corresponding values for our tree-based view manager (RBT) are 2.15 ms/cycle and 6.9 KB/cycle respectively, resulting in improvements (growing with group size) by a factor of $\approx$ 48. Note, that the manager scales up to groups consisting of 10K nodes showing constant server-side resource usage.

Figure 16b shows the results of our second test, where all group members join and leave periodically (every 15 seconds) resulting in a worst-case scenario. Despite the extremely high churn rate of roughly 330 joins / leaves per second, the tree-based view manager exhibits constant resource usage for up-to 5K nodes. Note, that a recent study on resource availability in Desktop Grids [6] shows that the mean host session time ranges between 2.8 hours for weekdays and 5.9 hours for weekends. Given these numbers, the CPU utilization for a 5K (1.51 ms/cycle) group managed by ORBWEB's tree-based view manager



(a) Unicast latency and throughput

(b) Groupcast latency and throughput

Fig. 17 Communication subsystem performance

would approximately range between 0.19‰ for weekdays and 0.09‰ for weekends on our quad-core server machine. These results substantiate that our solution supports a significant number of concurrent groups of considerable size for real-world churn rates.

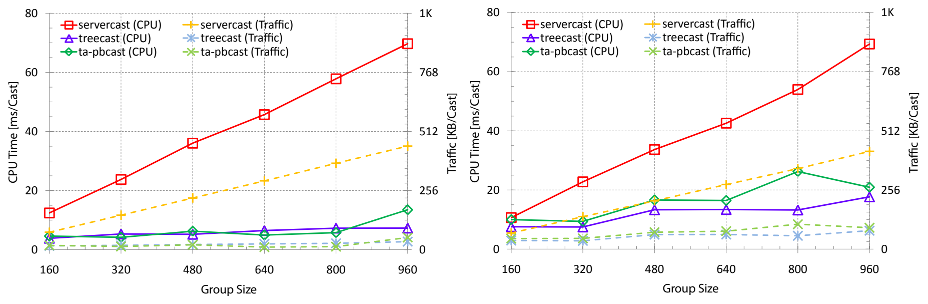
12.3 Communication

In this section, we report on the results of performance tests for the basic communication primitives.

12.3.1 I/O Subsystem Performance

With our first test we assess the performance of ORBWEB's XMPP processing pipeline, i.e. bandwidth and latency for unicasts and groupcasts with and without Fast Infoset encoding enabled. Our tests only address relayed communication, since E2E delivery of messages is based on plain TCP/IP. Note, that the test described herein is no scalability test for groupcasting which is addressed in the following section.

The bandwidth test is carried out by having the sender node emitting a large number of messages to a single receiver node (unicast) or to a group of size 1 (groupcast). In both cases, the single receiver acknowledges the receipt of all messages after the last message has arrived. Throughput is calculated as the quotient between transmitted payload bytes and the time elapsed between emission of the first message and receipt of the acknowledgment. Obviously, OPENFIRE/SMACK has problems in dealing with large messages (see Figures 17a and 17b). Due to a hardcoded server side limit on message size to prevent denial of service attacks, sending messages larger than 512K is impossible. For large messages between 16 KB and 512 KB our optimizations result in an improvement between 38% and 1038% for unicasts and between 130% and 1051% for groupcasts. With a resulting throughput of approximately 6.7 MB/s for unicast and 6.4 MB/s for groupcasts our substrate outperforms the reference implementation with 5.2 MB/s for unicasts and 5.6 MB/s for groupcasts. While our implementation is roughly on par with the reference implementation for small messages (i.e. [1 B, 256 B]), we achieve up to 29% improved throughput rates for unicasts and up to 13% for groupcasts when mid-size messages (i.e. [512 B, 8 KB]) are transmitted. This



(a) Non-volatile setup

(b) Volatile setup

Fig. 18 Server-side resource usage per groupcast operation

indicates that the dissection of the input stream as described in Section 9 adds no significant overhead to the parsing process.

Latency measurements are carried out in a ping-pong fashion. The sender emits a message that is echoed back by the recipient. The one-way latency is calculated as the time between emission of the message and receipt of the acknowledgment divided by two. Note that the latency numbers given here are actually two hop latencies (i.e. sender $\longrightarrow$ server $\longrightarrow$ receiver). Our optimizations result in very pronounced improvements for both unicasts and groupcasts and all messages sizes. Improvements in unicast latency are between 16% and 35% for small- and mid-size and 27% and 97% for large messages. The increases for groupcasts are between 21% and 48% for small- and mid-size and range from 28% to 99% for large messages. With 1.28 ms (unicast) and 1.36 ms (groupcasts), the minimal achievable latencies (i.e. for a message carrying no payload) are noticeably lower for our optimized implementation than for the reference implementation (1.53 ms and 2.65 ms respectively). The unicast latency is comparable to the latency of JXTA sockets [45] in the case of direct communication.

12.3.2 Groupcast Scalability

In this section, we assess the scalability and performance of three different groupcast implementations: the standard OPENFIRE/SMACK groupcast, and *ta-pbcast* with and without the anti-entropy phase enabled. The scenario consists of a master node groupcasting a message with 256 bytes payload five times a second within a static (Figure 18a) and a volatile (Figure 18b) group of variable size. In the volatile setup 12.5% of the nodes were configured to join and leave periodically, i.e. being offline for 30s and then online for 30s in an alternating manner.

The progressions of CPU utilization and bandwidth usage are linear with respect to the size of the group in the case of server-based groupcasting and are almost identical for both the static and the volatile setup. This is not surprising as node joins and departures cause virtually no overhead when the blind view manager is used. With 70 ms per groupcast for 960 nodes our quad-core server machine could ideally handle message rates of up-to 57 groupcasts per second. The costs per groupcast for treecast and ta-pbcast are almost constant at ≈ 5 ms per groupcast with a small linear overhead caused by the component detection that executes in parallel. This results in a maximum message rate of 571 groupcasts per second for tree-based groupcast and 296 groupcasts per second for ta-pbcast. The linear fraction

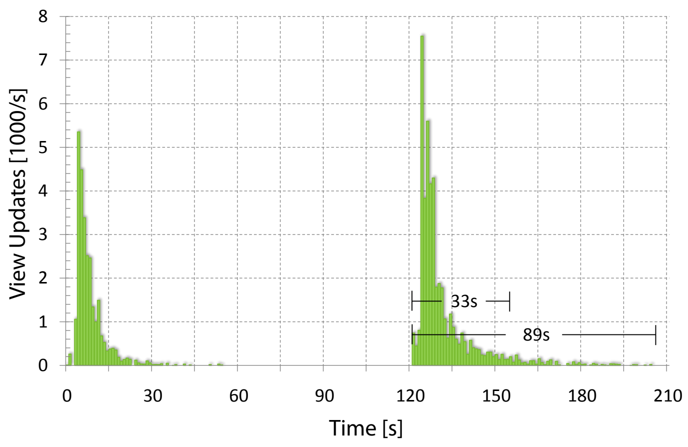


Fig. 19 Number of membership view updates emitted by the server on creation of a chord ring consisting of 640 nodes and on addition of 640 additional nodes. The latter takes 33s for the 95th percentile of updates and 89s until completion.

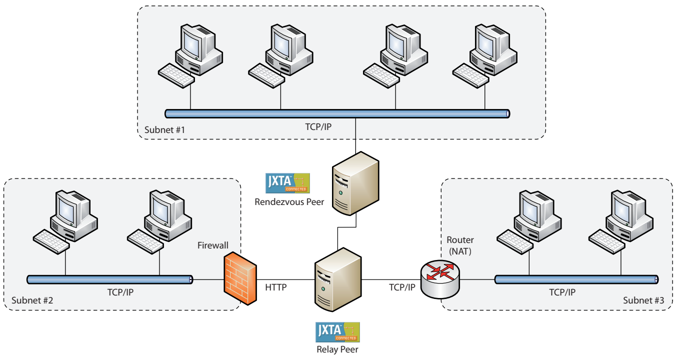
for component detection is more pronounced for the volatile setup as the constant flux in membership actually triggers E2E negotiations which is not the case in the static setup were possible session partners get greylisted quickly. Nevertheless, the costs for groupcasting are still significantly lower than that for the server-based groupcast. With 18 ms for the treecast and 21 ms for ta-pbcast maximum message rates are 222 and 190 groupcasts per second, respectively.

13 Related Work

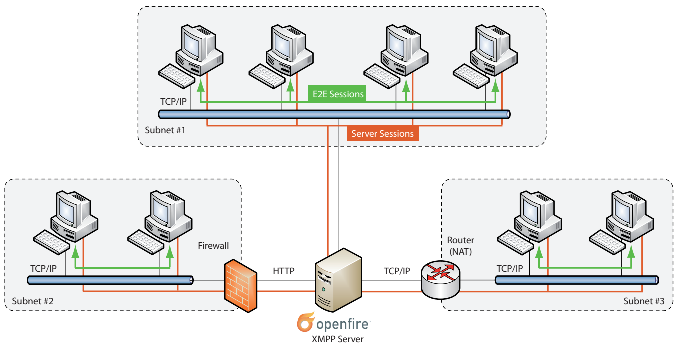
This section gives an overview of alternative technologies suitable for implementing (at least part of) the functionality provided by ORBWEB and justifies our decision to build a substrate for P2P Grid Computing based on XMPP by discussing a number of projects from the domain of distributed computing that have already migrated or will migrate to XMPP in the near future.

13.1 Alternative Substrate Technologies

During the last decade many structured P2P systems have been proposed [41,46,47,48,49] that implement packet routing and connection maintenance in a fully distributed way. Their goal is to deliver a packet with a particular destination address to the node closest to that address. This process typically involves forwarding the packet in a multi-hop fashion along the edges of an overlay network gradually approaching the destination node. Such overlay routing systems can be used to implement *Distributed Hash Tables* (DHTs). In principle, these systems can be used to implement part of ORBWEB's membership management functionality in a scalable way by employing the overlay neighbor sets to establish local membership lists. However, such a substrate would suffer from increased communication latency caused by multihop routing. As low latency is of prime importance in high performance computing, messages in our architecture are routed in a single hop for directly connected nodes,



(a) JXTA overlay network with rendezvous and relay infrastructure peers



(b) ORBWEB network with *E2E sessions* and an XMPP server

Fig. 20 JXTA / ORBWEB infrastructure comparison.

in two hops for nodes connected to the same XMPP server and in a maximum of three hops, i.e. Node #1 $\longrightarrow$ Server #1 $\longrightarrow$ Server #2 $\longrightarrow$ Node #2, for nodes connected to different XMPP servers. Furthermore, the churn resistance of overlay routing networks is limited as indicated by numerous evaluations conducted recently. For example, the routability of *BruNet* [48] drops to 84% when the mean session time falls below 6 minutes. Additionally, restoration of routability after massive node arrivals or departures happens comparatively slowly: *BruNet* needs 11 minutes to restore full routability after an insertion of 450 nodes to a fully routable network of 460 nodes [48]. Similar findings exist for Tapestry [46], which needs 10 minutes to restore 95% routability after inserting 200 nodes into an existing 325-node network. Although testbed setups are not identical, the fact that ORBWEB needs only 89 seconds to restore a perfect Chord ring after adding 640 nodes to an existing 640-node network (see Figure 19) indicates that using a server-asssisted DHT implementation is a promising approach, when performance is considered more important than outmost scalability. Note, that routability virtually remains unaffected as long as the XMPP server is not overloaded. This results from the fact that nodes are inserted into the ring right after they joined the group.

JXTA [25] is an open source initiative, sparked and maintained by Sun Microsystems™. It's primary goal is to provide a foundation for interoperable P2P applications. JXTA consists of a set of six language- and platform-independent protocol specifications. It provides basic services for generic P2P applications including peer group organization, inter-peer communication, and resource discovery. Note, that JXTA is neither limited to nor optimized for P2P Grid Computing. While it provides higher-level services like discovery and monitoring, it lacks explicit support for membership views, which we consider fundamental for high performance computing. However, both JXTA and XMPP provide similar functionality on the lower protocol layers dedicated to communication. Although JXTA is the most advanced P2P library currently available and is considered to be a good choice for implementing distributed computing platforms [45], performance studies have revealed weaknesses in the Java™ implementation of the version 2.0 protocol specification. That includes pipe latency and throughput for small messages [50], reliability of TCP connections [51], and rendezvous network stability [52]. With the addition of E2E sessions, the ORBWEB network infrastructure becomes very similar to the infrastructure of the JXTA [25] network (see Figure 20): The XMPP server is, like the *rendezvous peer* in JXTA, responsible for delivering groupcast messages to all connected users. Unicast messages are delivered over direct connections between clients. If no direct connection is possible due to NAT devices or firewalls, the XMPP server delivers the message conventionally. This is similar to the responsibility of *relay peers* in the JXTA network architecture.

Peer-to-Peer Simplified (P2PS) [53] is an open-source project providing an infrastructure for P2P service discovery and pipe-based communication. The P2PS reference implementation is written in Java™. While sharing many concepts with JXTA, P2PS is more focused on simplicity than on feature richness. P2PS peers can communicate over multiple protocols that can be replaced transparently to the application. P2PS service discovery is based on XML advertisements and queries and uses subnets to broadcast advertisements and queries efficiently. As in JXTA, rendezvous peers are responsible for caching and forwarding advertisements and queries to rendezvous' in other subnets. Although P2PS provides a peer group abstraction, there is no support for broadcasting within groups. Thus, a substantial requirement for many distributed algorithms is not satisfied. To our knowledge there is no performance evaluation available for P2PS.

13.2 XMPP Technology Adopters

OurGrid [54] is a platform for P2P Grid Computing. Currently (version 3.3), OurGrid uses *Remote Method Invocation* (RMI) over TCP/IP as programming model. RMI suffers from poor performance, due to a large protocol overhead and also does not integrate well with restrictive firewalls and NAT devices. In the upcoming 4.0 release, OurGrid will switch to an asynchronous programming model called JDIC [55] that employs XMPP as transport protocol.

The *Distributed Infrastructure with Remote Agent Control* (Dirac) [56] is a Service Oriented Architecture (SOA) composed of lightweight services forming a scalable robust Grid Computing environment to manage and track a large number of computing tasks. Since Dirac is primarily targeted for high-throughput computing, it makes not as high demands on the communication infrastructure as P2P Grid Computing focused on high-performance computing does. XMPP is used to implement three different aspects of the system: inter-service messaging, state monitoring of agents, and job-level monitoring. While plain XMPP is suitable for the first two applications as the number of services (5-20) and agents (10-100) is comparatively low, the third is more critical as thousands of jobs are active at peak time. In contrast to our approach of improving and extending the XMPP protocols, Dirac restricts itself to protect the system from the impact of overloaded XMPP servers by applying virtualization techniques. Certainly, Dirac could benefit from adopting our improved network substrate.

Xeerkat [57] is a Grid economy platform based on dynamically reconfigurable networks of agents that offer and consume services. Due to better infrastructure support and much easier setup [58] Xeerkat migrated from JXTA to XMPP in the 2.0 release.

The *Friend-to-Friend Computing framework* [59] project envisions to enable what the authors call *F2F Grids* or *Frids*. The key distinctive feature of Frids in comparison to Grids is their ease of use: they allow to start a parallel application quickly without any administrative effort. To finally fulfill the initial vision of Grid Computing being as easy to use as the power grid, the authors propose to use XMPP as the enabling communication and coordination platform. Although Frids are pretty simple to setup by exploiting existing instant messaging infrastructure, their usability for solving demanding computational problems is questionable and has not been evaluated on a large-scale.

14 Conclusion

Supporting non-trivial parallelism for Desktop Grid Computing requires efficient P2P interaction among the nodes. This characteristic feature must ultimately be enabled by the network substrate technology. Such a substrate must satisfy a set of functional (i.e. membership management and fundamental communication primitives) and non-functional requirements (i.e. performance, scalability, connectivity, and security) to be able to serve as a basic building block for existing and future P2P Grid Computing platforms. In this paper, we demonstrate that a substrate based on the XMPP standard can fulfill all these functional and non-functional requirements.

We demonstrate, how to substantially improve the scalability of an industrial strength XMPP stack (OPENFIRE/SMACK) by (i) leveraging Jingle as a mechanism to transparently establish direct P2P links and by (ii) modifying the XMPP MUC protocol and component to support fully customizable membership views that among other things allow for the creation of highly scalable tree-based topologies. By (iii) adopting and adapting a robust probabilistic multicast protocol, we were able to significantly reduce the server-side load caused by groupcast processing. Finally, we have shown, (iii) how to achieve significantly better communication performance by using a binary XML encoding.

To prove the applicability of ORBWEB as a network substrate for large-scale distributed systems in general and for P2P Grid Computing systems in particular, we described how to implement custom view managers for sophisticated higher-level services and conducted a detailed evaluation that confirmed that our optimizations in total yield substantial performance and scalability improvements even under massive churn rates.

Currently, we are researching on how to distribute groups across a self-organizing network of ORBWEB servers that are created and destroyed on-demand. Such an extension will further push ORBWEB's scalability beyond 10K nodes.

Acknowledgements

Sven Schulz and Mathias Poths are supported by Deutsche Forschungsgemeinschaft (DFG) under grant BL 941/1-2. We are grateful to Hannes Hannak and Thomas Rösner for sharing their expertise in testbed setup and maintenance.

References

1. Sven Schulz, Wolfgang Blochinger, Markus Held, and Clemens Dangelmayr. COHESION - A microkernel based desktop grid platform for irregular task-parallel applications. *Future Generation Computer Systems – The International Journal of Grid Computing: Theory, Methods and Applications*, 24(5):354–370, 2008.
2. Sven Schulz and Wolfgang Blochinger. An integrated approach for managing peer-to-peer desktop grid systems. In *Proc. of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, pages 233–240, Rio de Janeiro, Brazil, May 2007.
3. Wolfgang Blochinger, Wolfgang Westje, Wolfgang Küchlin, and Sebastian Wedeniwski. ZetaSAT – Boolean satisfiability solving on desktop grids. In *Proc. of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, volume 2, pages 1079–1086, Cardiff, UK, May 2005.
4. Wolfgang Blochinger, Clemens Dangelmayr, and Sven Schulz. Aspect-oriented parallel discrete optimization on the cohesion desktop grid platform. In *Proc. of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*, pages 49–56, Singapore, May 2006.
5. Sven Schulz, Wolfgang Blochinger, and Mathias Poths. A network substrate for peer-to-peer grid computing beyond embarrassingly parallel applications. In *Proc. of International Conference on Communications and Mobile Computing (CMC 2009)*, Kunming, Yunnan, China, January 2009, accepted. IEEE Computer Society.
6. Derrick Kondo, Michela Taufer, Charles L. Brooks, Henri Casanova, and Andrew A. Chien. Characterizing and evaluating desktop grids: An empirical study. In *Proc. of International Parallel and Distributed Processing Symposium*, Sante Fe, New Mexico, 2004.
7. David P. Anderson and Gilles Fedak. The computational and storage potential of volunteer computing. In *Proc. of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*, pages 73–80, Singapore, 2006.
8. David P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proc. of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Pittsburgh, USA, 2004.
9. Andrew Chien, Brad Calder, Stephen Elbert, and Karan Bhatia. Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel Distributed Computing*, 63:597–610, 2003.
10. Yoshio Tanaka Kazuyuki Shudo and Satoshi Sekiguchi. P3: P2P-based middleware enabling transfer and aggregation of computational resources. In *Proc. Cluster Computing and Grid 2005 (Fifth Int'l Workshop on Global and Peer-to-Peer Computing)*, Cardiff, UK, 2005.
11. Jerome Verbeke, Neelakanth Nadgir, Greg Ruetsch, and Ilya Sharapov. Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. In *Proceedings of the Third International Workshop on Grid Computing (GRID '02)*, pages 1–12, London, UK, 2002. Springer-Verlag.
12. Ian Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *International workshop on peer-to-peer systems (IPTPS 2003)*, Berkeley, CA, USA, 2003.
13. Derrick Kondo, Gilles Fedak, Franck Cappello, Andrew A. Chien, and Henri Casanova. On resource volatility in enterprise desktop grids. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 78, Washington, DC, USA, 2006. IEEE Computer Society.
14. The XMPP Software Foundation. http://www.xmpp.org (04/24/2008).
15. P. Saint-Andre. End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP). RFC 3923 (Proposed Standard), October 2004.
16. ejabberd - The Erlang Jabber/XMPP daemon community site. http://www.ejabberd.im (04/24/2008).
17. Ignite Realtime - A Jive Software Community. http://www.igniterealtime.org (04/24/2008).
18. Fast Infoset Project. http://fi.dev.java.net (04/24/2008).
19. MINA. http://mina.apache.org (04/24/2008).
20. Ayalvadi J. Ganesh, Anne-Marie Kermarrec, and Laurent Massoulié. Peer-to-Peer Membership Management for Gossip-based Protocols. *IEEE Trans. Comput.*, 52(2):139–149, 2003.
21. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, January 1995.
22. Sven Schulz, Wolfgang Blochinger, and Hannes Hannak. Capability-aware information aggregation in peer-to-peer grids - methods, architecture, and implementation. *Journal of Grid Computing*, 2008. Submitted, http://www.cohesion.de/cms/fileadmin/publications/aggregation.jogc2008.pdf.
23. Dale Skeen and Michael Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, pages 295–317, 1987.
24. E. W. Dijkstra, W. H. J. Feijen, and A. J. M. van Gasteren. Derivation of a termination detection algorithm for distributed computations. In *Proc. of the NATO Advanced Study Institute on Control flow and data flow: concepts of distributed programming*, pages 507–512, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

25. Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. Project JXTA 2.0 Super-Peer Virtual Network. Technical report, Sun Microsystems, May 2003.
26. Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.
27. Öznur Özkasap and Kenneth P. Birman. Throughput stability of reliable multicast protocols. In *ADVIS '00: Proceedings of the First International Conference on Advances in Information Systems*, pages 159–169, London, UK, 2000. Springer-Verlag.
28. David R. Cheriton and Dale Skeen. Understanding the limitations of causally and totally ordered communication. *SIGOPS Oper. Syst. Rev.*, 27(5):44–57, 1993.
29. Rico Piantoni and Constantin Stancescu. Implementing the swiss exchange trading system. In *FTCS '97: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, page 309, Washington, DC, USA, 1997. IEEE Computer Society.
30. Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. Netw.*, 5(6):784–803, 1997.
31. Ching-Gung Liu. *Error recovery in scalable reliable multicast*. PhD thesis, University of Southern California, Los Angeles, CA, USA, 1997.
32. Matthew Thomas Lucas. *Efficient data distribution in large-scale multicast networks*. PhD thesis, Pennsylvania State University, 1998.
33. James O. Coplien and Douglas C. Schmidt, editors. *Pattern Languages of Program Design*, chapter Reactor: an object behavioral pattern for concurrent event demultiplexing and event handler dispatching, pages 529–545. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.
34. Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, 1999.
35. Rob V. van Nieuwpoort, Thilo Kielmann, and Henri E. Bal. Efficient load balancing for wide-area divide-and-conquer applications. In *PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 34–43, New York, NY, USA, 2001. ACM.
36. Christos Gkantsidis, Milena Mihail, and Amin Saberi. Hybrid search schemes for unstructured peer-to-peer networks. In *INFOCOM*, pages 1526–1537. IEEE, 2005.
37. Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.
38. Arturo Crespo and Hector Garcia-Molina. Routing indices for peer-to-peer systems. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 23, Washington, DC, USA, 2002. IEEE Computer Society.
39. Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range queries in trie-structured overlays. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 57–66, Washington, DC, USA, 2005. IEEE Computer Society.
40. S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Range queries over DHTs. Technical Report IRB-TR-03-009, Intel Research, Berkley, 2003.
41. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.
42. Praveen Yalagandula and Mike Dahlin. A scalable distributed information management system. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 379–390, New York, NY, USA, 2004. ACM.
43. Roland Wiese, Markus Eiglsperger, and Michael Kaufmann. yfiles: Visualization and automatic layout of graphs. In *Graph Drawing*, pages 453–454, 2001.
44. P. Gomez and P. Aston. The Grinder V3.0, July 2008. http://grinder.sourceforge.net/.
45. G. Antoniu, P. Hatcher, M. Jan, and D. A. Noblet. Performance evaluation of jxta communication layers. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pages 251–258, Washington, DC, USA, 2005. IEEE Computer Society.
46. Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications*, pages 41–53, 2003.
47. Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

48. P. Oscar Boykin, Jesse S. A. Bridgewater, Joseph S. Kong, Kamen M. Lozev, Behnam Attaran Rezaei, and Vwani P. Roychowdhury. A symphony conducted by brunet. *Computing Research Repository (CoRR)*, arXiv:0709.4048v1, 2007.
49. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.
50. Emir Halepovic and Ralph Deters. The costs of using jxta. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 160, Washington, DC, USA, 2003. IEEE Computer Society.
51. Jean-Mark Seigneur. JXTA Pipe Performance. http://bench.jxta.org (04/24/2008).
52. K. Burbeck, D. Garpe, and S. Nadjm-Tehrani. Scale-up and performance studies of three agent platforms. In *IEEE International Conference on Performance, Computing, and Communications*, pages 857– 863, 2004.
53. Ian Wang. P2PS (Peer-to-Peer Simplified). In *Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies*, pages 54–59. Louisiana State University, February 2005.
54. Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the World, Unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006.
55. Aliandro Lima, Walfredo Cirne, Francisco Vilar Brasileiro, and Daniel Fireman. A Case for Event-Driven Distributed Objects. In *OTM Conferences (2)*, pages 1705–1721, 2006.
56. Andrei Tsaregorodtsev, Vincent Garonne, and Ian Stokes-Rees. Dirac: A scalable lightweight architecture for high throughput computing. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 19–25, Washington, DC, USA, 2004. IEEE Computer Society.
57. R. A. Milowski. Computing for the mathematical sciences with xml web services and p2p. In *XML 2005*, Atlanta, Georgia, U.S.A., November 2005.
58. R. A. Milowski. Xeerkat - A P2P computing framework over XMPP. http://code.google.com/p/xeerkat/ (8/18/2008).
59. Ulrich Norbisrath, Keio Kraaner, Eero Vainikko, and Oleg Batrasev. Friend-to-friend computing - instant messaging based spontaneous desktop grid. In *Third International Conference on Internet and Web Applications and Services*, pages 245–256, Los Alamitos, CA, USA, 2008. IEEE Computer Society.