Quality of Service for Peer-to-Peer based Networked Virtual Environments

Marc Brogle, Dragan Milić, Torsten Braun
Institute of Computer Science and Applied Mathematics
Universität Bern, Neubrückstrasse 10, 3012 Bern, Switzerland
brogle@iam.unibe.ch, milic@iam.unibe.ch, braun@iam.unibe.ch

Abstract

This paper describes how Quality of Service (QoS) enabled Overlay Multicast architectures using Peer-to-Peer (P2P) networks can enhance the experience of end-users in Networked Virtual Environments (NVE). We show how IP Multicast, which offers an easy to use API for implementing NVE but is not widely deployed, can be made available to end-users by bridging it transparently with P2P networks. We describe how different P2P and Application Level Multicast (ALM) architectures can be extended with QoS mechanisms using our proposed OM-QoS (Overlay Multicast QoS) architecture. The presented approach allows users to experience QoS for NVE such as group-based multimedia broadcasting and distributed multiplayer games.

1 Introduction

Efficient data dissemination for Networked Virtual Environments (NVE) such as group-based multimedia broadcasting or data exchange in distributed and de-centrally-managed multiplayer games can be realized using multicast. Multicast senders transmit their data/stream only once while the network or other participants duplicate and deliver the data to subscribers. This allows to create efficient, scalable and distributed group-based NVE that require no central management or expensive (server-based) infrastructure. IP Multicast unfortunately has not been deployed widely in the Internet today and is not really usable by end-users. Multicasting can though be made available to end-users by using Application Level Multicast (ALM) running on top of Peer-to-Peer[6] (P2P) networks. Unfortunately, no real standard for ALM (contrary to the IP Multicast API) exists. We present an approach to enable IP Multicast usage for end-users by bridging IP Multicast interface with ALM for data distribution and describe an architecture to enable Quality of Service (QoS) in P2P/ALM networks. This allows NVE to apply the easy to use IP Multicast API with QoS mechanisms to improve the user's experience.

The remainder of the paper is organized as follows. The next Section describes how IP Multicast can be bridged with ALM. Section 3 describes the basic properties of the OM-QoS (Overlay Multicast QoS) architecture. In Section 4 we apply OM-QoS to different P2P/ALM architectures. A short evaluation follows in Section 5. Finally, a conclusion and outlook are given in Section 6.

2 Bridging IP Multicast and ALM

In [7] we presented how IP Multicast can be made available to end-users by using a virtual network interface and an Application Layer Multicast (ALM). The Multicast Middleware, developed for the EuQoS[5] IST project in the European 6th framework program, acts as an IP Multicast router attached to a virtual network interface.

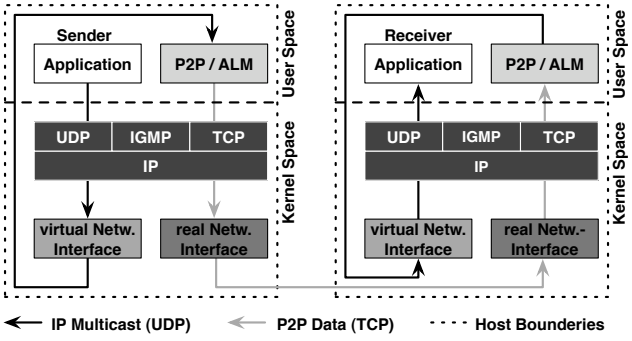

Figure 1. Bridging IP Multicast and ALM

Figure 1 shows how the Multicast Middleware can capture IP Multicast packets that are routed by the operating system through the virtual network interface (network TAP[13]) and afterwards the packets will not leave the end-system via the network. Instead, these packets are disseminated using a P2P/ALM infrastructure (e. g. Scribe/Pastry). IGMP messages trigger the setup of the ALM. At the receiver, the IP Multicast traffic is injected back into the system using TAP. This is completely transparent to applications, existing IP Multicast enabled applications do not need to be changed.

3 OM-QoS basic principles

The basic idea of OM-QoS is to build QoS-aware multicast trees as presented in [2]. Therefore the tree construction mechanisms of the investigated P2P/ALM should yield in a multicast tree as shown in Figure 2.

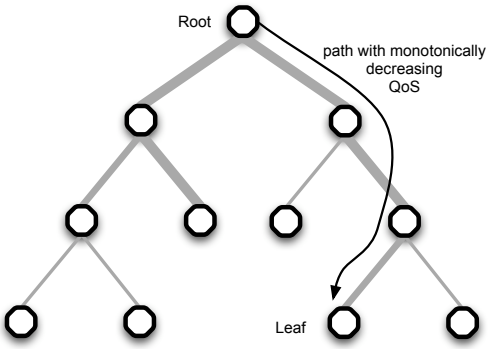

Figure 2. OM-QoS Multicast Tree setup

Such a tree should hold the basic property: *"All paths from the root to the leafs have monotonically decreasing QoS."* The QoS classes used to distinguish the different quality requirements have to satisfy the following three requirements: (1) all QoS classes have a total order relation, (2) QoS class parameters are independent of link length and number of hops in the network, and (3) the amount of QoS classes is finite. An example QoS class would be the bandwidth or jitter, but not delay, which is additive and depends on link length and hop count. Also combinations of some parameters could be used, for example bandwidth and jitter together, as long as it holds the three properties described.
Most P2P networks already minimize the delay between the participating peers. In order to fully support delay-related QoS requirements, the end-to-end delay from the root to a leaf over the in-between hops along the path has to be considered (sum of hop delays). We plan to support that end-to-end delay related QoS parameter as well. It will be treated as a (from the presented QoS class concept) independent and separate parameter, which will be taken into account when looking for a potential parent in the P2P network.
At the moment we only consider network provided QoS, users get QoS guarantees provided by the underlying network infrastructure. As a next step we will also look at dynamic environments using measurement based best-effort QoS, where trees have to adapted periodically if QoS fails.
OM-QoS aims to be a general solution for introducing QoS capabilities to different structured P2P/ALM architectures and is not limited to DHT like systems. If a P2P/ALM protocol cannot be modified to support QoS as done for Scribe/Pastry, Bayeux/Tapestry and NICE, the layered approach (used for CAN) can be applied as a general solution.

4 Applying OM-QoS to P2P/ALM

4.1 Scribe/Pastry

Pastry[11] is a P2P routing substrate with a ring structure (one-dimensional torus). Peers chose a random (uniformly distributed) and unique ID when they join the P2P network. Routing in Pastry uses Plaxton's method[9]. Each hop from source to destination matches one or more prefixes of the message's destination address. A peer has more information about ID-neighbors (matching many prefixes of the ID left to right) than about ID-distant peers matching less ID prefixes (left-to-right). Routing is proximity aware: each hop from source to destination tries to minimize the intermediate hop-delay. Therefore, the overall end-to-end delay can be optimized up to a certain degree. Figure 3 (a) shows a simplified example of Pastry's prefix matching routing mechanism. In the example we want to route a message from the source **BCD** to the destination **EDE**. The source only knows the node **EAA**, which shares the first prefix with the destination address **EDE**. The message is forwarded (1) to this node, which forwards (2) the message to **EDC** matching the next prefix of the destination address in its own routing table. Finally, node **EDC** delivers (3) the message to the destination **EDE**, which it knows directly. If there is no node, with the ID of a message's destination address assigned, the node with the ID numerically closest to the destination address is responsible for message message.
Scribe[4] is a core based ALM infrastructure, running on top of Pastry. Each multicast group (called topic) has a unique multicast group (topic ID), which has a core (root) node for the distribution tree. This root is the node with the Pastry ID numerically closest to the topic ID. All multicast traffic for that group is forwarded to the root node for dissemination. Figure 3 (b) shows a simplified example of a multicast tree construction with Scribe. A joining node sends a *join* message to the topic's root node using Pastry routing. Nodes on intermediate hops along the path of the *join* message add the previous node/hop to the list of receivers for that topic. A *join* message is only forwarded further towards the root if the current intermediate hop is not yet subscribed to that topic. When data for that topic ID has to be disseminated, the root node forwards the message to all its one-hop subscribed nodes. These nodes repeat the same process, forwarding the message to their one-hop subscribers. In the example node **BCD** wants to join topic with ID **EDE**. The same intermediate hops are visited as in the Pastry example and the multicast tree distribution tree for this subscriber is therefore built using the reverse path of the *join* message. Other joining hosts (e. g. **AFB** and **DAB**) could send their join message via nodes that are already subscribed to the topic (e. g. **EDC** and **EAA**), which then stop forwarding the *join* message towards the root.

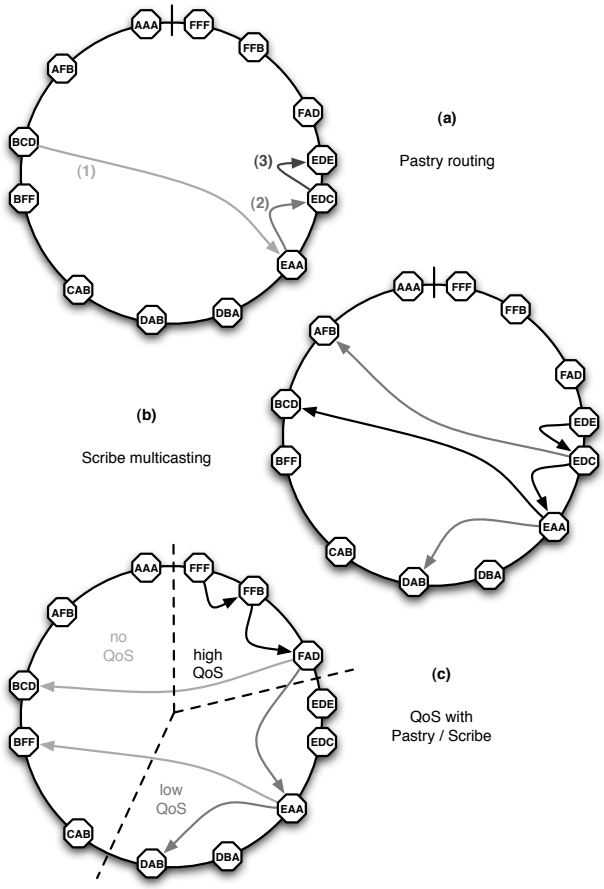

Figure 3. Pastry / Scribe

To enable QoS for Scribe/Pastry and enforce the creation of QoS aware multicast trees (see Section 3), the following modifications of Pastry's ID assignment method have to be performed: (1) a dedicated Pastry network for each active multicast group exists (eliminates not interested forwarders), (2) higher QoS requirements of a peer, result in higher Pastry IDs, and (3) all peers subscribe to the highest possible topic ID (node with the highest QoS requirements will be root). These three design choices ensure that all the paths from the root to the leafs will have monotonically decreasing QoS requirements. Figure 3 (c) shows such an example where all paths from the root node **FFF** to the leafs will on each hop either pass through hops with the same or lower QoS requirements than on the previous hops. In this example there are three different QoS classes, which are not distributed evenly among the ID space. The QoS class segments have to be assigned in such a way that for each higher QoS class at least one additional digit (from left to right) matches the topic's ID compared to the previous lower QoS class. A more detailed description and evaluation of this approach has been presented in [2]. There we also showed that basic properties of Scribe/Pastry such as the average end-to-end path length are not changed significantly by introducing the previously mentioned modifications, even though the ID distribution does not follow anymore strictly a normal distribution.

4.2 Tapestry / Bayeux

Tapestry[14] is similar to Pastry as described in Section 4.1 and uses also prefix routing as described before with randomly assigned IDs. The prefixes are matched from right-to-left, whereas Pastry uses left-to-right prefix matching. Figure 4 (a) shows a simplified example of Tapestry's routing mechanism, where a message is sent from node **8311** to node **4985** and on each hop one additional prefix is matched (xxx5 → xx85 → x985 → 4985).

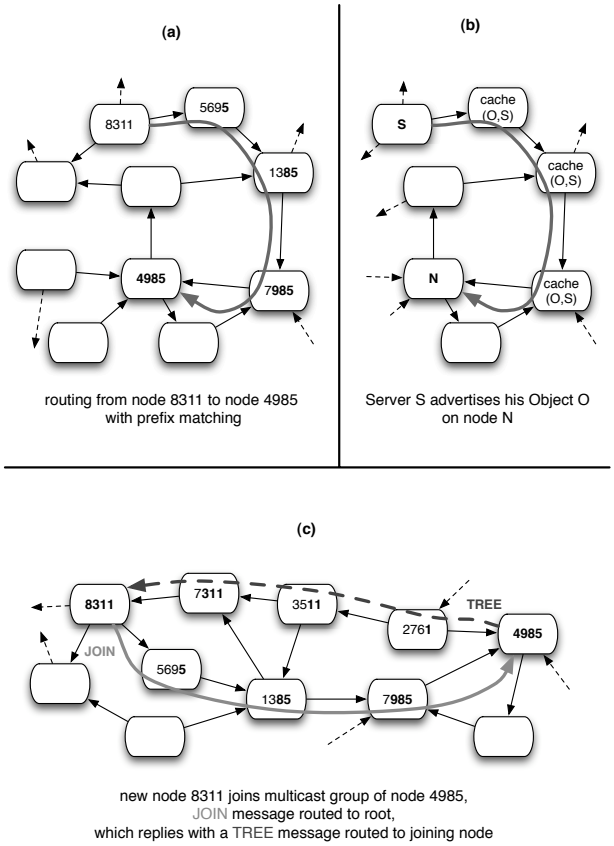

Figure 4. Tapestry and Bayeux

Tapestry is used for distributed data storage; an example is shown in Figure 4 (b). A data object O stored at S is advertised sending a *publish* message to N (whose $ID = hash(O)$) using Tapestry's routing. Intermediate nodes on the path cache this advertisement. Nodes requiring the lo-

cation information of object O can find and contact N using the hash function $hash(O)$ but might already get an answer from a node on the path to N caching the requested information. Caching reduces the load on the lookup node N.
Bayeux[15] is a source-specific, explicit join multicast facility, which runs on top of Tapestry. Figure 4 (c) shows a simplified example of Bayeux's multicast tree creation. The root for a multicast group advertises, using a *publish* message, that it is the responsible node for this multicast group. Joining nodes send a *join* message to the root. The root node answers with a *tree* message. Each node on the path from the root to the joining node (for the *tree* message) saves the forwarding state $< dest, nexthop >$. The *join* messages are always delivered to the root node, which results in higher link stress on the root compared to Pastry. The multicast delivery path (constructed by the *tree* message) is not the reverse-path of the *join* message (as for Scribe).
To enable QoS aware multicast tree setup with Bayeux, a similar mechanism can be used as with Scribe. Due to the different prefix matching order and the non-reverse setup of the paths, the approach has to be slightly modified compared to Scribe/Pastry as follows: to ensure that all paths from the root to any group member hold the basic property described in Section 3, higher QoS requirements of a peer result in more digits matching the root's ID (right to left). We still use a dedicated P2P network per multicast group.

4.3 NICE / ZigZag

In NICE[1] multicast group members are arranged hierarchically. Hosts are grouped together in clusters with a predefined maximum cluster size. Hosts are clustered by comparing the round-trip time (RTT). Close together hosts (in terms of RTT) are put in the same cluster. Each cluster has a cluster leader, which is determined by calculating the graph-theoretical RTT center. Hosts in the same cluster are called cluster mates. After all hosts are assigned to a cluster and a cluster leader has been determined for each cluster, a new round of clustering is performed among these cluster leaders on a new layer. This is continued until there is only one cluster left. The different cluster groups are therefore arranged in layers as shown in the example in Figure 5 (a). Cluster leaders multicast data among all their cluster mates in all their clusters. A cluster leader, which would be a member of a cluster in each layer, would have a very high fan out (depending on the cluster size). Note that for n hosts with a maximum cluster size of s, the resulting amount of layers would be $\log_s(n)$. Therefore a cluster leader could have to serve up to $s \log_s(n)$ hosts with the multicast data. The authors of NICE therefore state that it has not been designed for high bandwidth data transmission.
To reduce the high fan-out ZigZag[12] was introduced. Cluster mates in ZigZag only receive messages from foreign

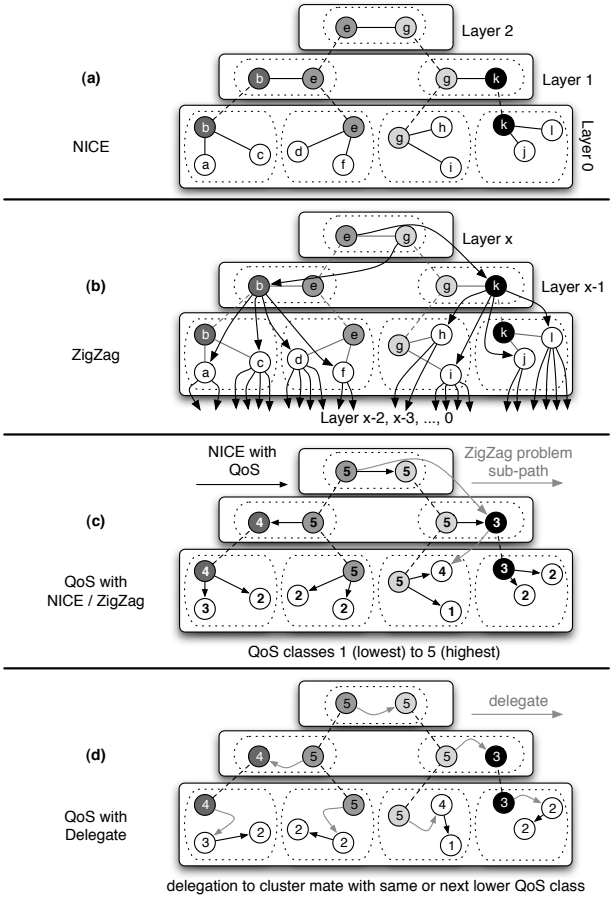

Figure 5. NICE / ZigZag / Delegate

cluster heads, which send only to one layer below them. A cluster leader *h* for a host *x* has in the next higher layer a cluster mate *m*, which is a foreign cluster head for host *x*. As Figure 5 (b) shows, the fan-out for a cluster head can be reduced using ZigZag. A cluster leader now only has to serve up to $2s-2$ other hosts, but typically only $s-1$ other hosts for a maximum cluster size s. The fan-out of a cluster leader is now independent of the numbers of layers.
To introduce QoS to NICE, we had to take a completely different approach, but at the end only the cluster leader determining mechanism has to be modified: the cluster leader is determined by the highest QoS class inside a cluster (and not the graph-theoretic center). An example is shown in Figure 5 (c). Clusters are still built using RTT measurements, and we have 5 QoS classes, with QoS class 5 as the highest one. Cluster leaders are now hosts having the highest QoS class in the cluster. Figure 5 (c) also shows that the proposed modification does not work with ZigZag, because foreign cluster heads can have lower QoS classes than some of the hosts in the cluster to which they act as foreign head. To reduce the fan-out with QoS support, a delegation mech-

anism (adding an additional hop) can be used: cluster leaders delegate message dissemination for each of their clusters to cluster mates with same or next lower QoS class as themselves. This results in a fan-out up to $\log_s(n)$ for a cluster leader (s = max. cluster size, n = amount of hosts). Figure 5 (d) shows an example with the delegation mechanism.

4.4 CAN

Content Addressable Networks (CAN)[10] use a virtual d-dimensional Cartesian coordinate space to store key - value pairs. The space is partitioned into *n* (number of hosts) zones, which correspond to the keys. Each node is responsible for managing one part of the coordinate space. The coordinate space has to be rearranged if new nodes join, existing zones have to be divided between the new and existing nodes. Changes of zones are propagated to neighbors (adjacent zones), who update their neighbor sets. Neighbor sets contain all information about adjacent zones for a host. Routing is greedy using the neighbor closest to the destination as next hop. Multicast is done by sending a message to all the neighbors of a host with duplicate suppressing.
The example in Figure 6 (a) shows a CAN with 5 nodes and 5 zones. In Figure 6 (b) routing is presented. Node 5 routes a message to node 1, using node 6 as next hop, which uses then its neighbor 3 to reach the destination 1. Figure 6 (c) shows what happens when a new node joins. The new node 9 assigns itself random coordinates, which would e. g. fall into the zone of node 5. Node 5 splits its own zone and gives half of the space to the newly joined node. The neighbor set of node 5 has then to be changed and node 9 builds its own neighbor set. Nodes 5 and 9 then inform their neighbors.
Introducing QoS to CAN by mapping QoS classes to initial coordinates does not work, because the location (zones) of a node can change overtime (zones for nodes move). Figure 6 (d) shows an example where this approach fails. Higher QoS classes result in higher initial coordinates. Splitting zones can lead to have hosts with higher QoS (initial coordinates) being positioned "below" hosts with lower QoS coordinates. Node 4 (with QoS 0.4,0.4) is placed "below" node 1 (with QoS 0.2,0.2). The property described in Section 3 does not hold: the QoS requirements are not monotonically decreasing for a message being routed from the root (highest QoS = highest coordinates) to node 4.
To enable QoS for CAN a layered approach has been chosen. An example with three QoS classes is shown in Figure 7. Each QoS class has a dedicated CAN. Hosts having the same QoS class join the same CAN. Senders **S** disseminate the multicast data inside their own CAN and then send the data to the CAN with the next lower and next higher QoS class. Nodes receiving multicast messages from another CAN disseminate them in their own CAN and send them to the CAN with next higher or lower QoS class depending on the initial "direction" (up or down). Delay can be further reduced by sending messages further up/down in the layer hierarchy instead of only to adjacent layers.

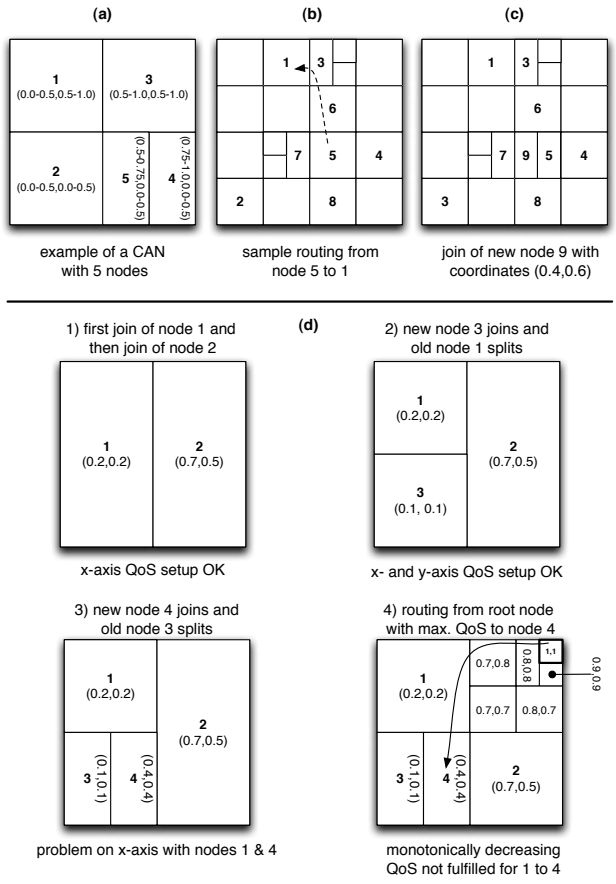

Figure 6. CAN routing and QoS problems

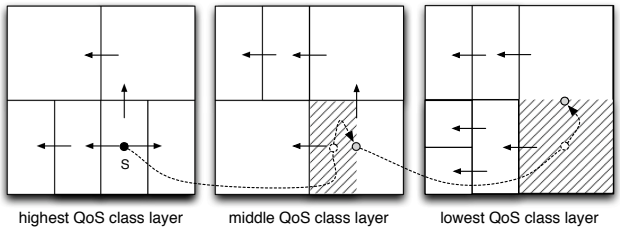

Figure 7. QoS for CAN with layers

5 Evaluation

We used the freely available implementation of Scribe/Pastry called Freepastry, for the ALM used by the Multicast Middleware and applied also the OM-QoS architecture to the package. The Multicast Middleware runs on

various operating systems (Win32, Mac OS X, Linux). We evaluated QoS behavior in [2], where we used the simulator that comes bundled with Freepastry. There we showed that our modification yields in 100% of the paths holding the mentioned three basic properties, whereas the random ID assignment of Pastry roughly lead to an average of 40% of the paths holding those three basic properties.
Latency and bandwidth are important issues for NVE: data in multiplayer games should be forwarded to nearby players quickly. Multimedia applications require high bandwidth. Our measurements performed in [3] show that the Multicast Middleware can process data rates up to 155 Mbps (on Pentium D 3GHz CPU based PCs running Linux) with an acceptable packet loss rate and no dramatic jitter increase. Recent measurements showed that end-to-end delay for packet sizes of 512 & 1024 bytes sent at rates up to 75.2Mbps were generally below 10ms for various chain/tree topologies built with 7 PCs (Pentium IV 3GHz running Linux) in a LAN.

6 Conclusion and Outlook

In this paper we presented how the Multicast Middleware and OM-QoS can enhance the experience of end-users in Networked Virtual Environments (NVE). Multimedia broadcasting and distributed de-centrally managed multiplayer games can profit using QoS enhanced IP Multicast. Our approach has been successfully implemented and tested in the European research project EuQoS. It offers high bandwidth support, intelligent delay-optimized tree creation, low latency while processing the exchanged data, and is completely transparent to applications. We presented different mechanisms to introduce QoS to P2P/ALM networks. For Scribe/Pastry we apply the OM-QoS mechanism first presented in [2]. That idea has now been further analyzed and extended to make it applicable for other protocols such as Bayeux/Tapestry and NICE, which works completely different than Scribe/Pastry. We also enhanced OM-QoS to support almost any kind of P2P/ALM architecture by introducing a layered approach (used for CAN) as a general solution, also for unstructured P2P networks.
The different concepts presented in this paper to enable QoS for P2P/ALM networks still need further investigation. We are implementing OM-QoS (protocol specific and layered approach) for several P2P/ALM systems in the OMNET++[8] simulator. Supporting QoS for CAN using multiple dimensions could be analyzed further. End-to-end delay guarantees will also be further examined and integrated as a separate independent QoS parameter. Most investigated P2P networks optimize delay between peers and we try to keep this property when applying OM-QoS to those protocols. We will not only analyze OM-QoS with network provided QoS guaranties (hard QoS) but also in dynamic environments using measurement based (soft) QoS.

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *SIGCOMM02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 32, pages 205–217, New York,USA, 2002. ACM.

[2] M. Brogle, D. Milic, and T. Braun. QoS enabled multicast for structured P2P networks. In *Workshop on Peer-to-Peer Multicasting at the 4th IEEE Consumer Communications and Networking Conference*. IEEE, January 2007.

[3] M. Brogle, D. Milic, and T. Braun. Supporting IP multicast streaming using overlay networks. In *QShine: International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*. ACM, August 2007.

[4] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *Selected Areas in Communications, IEEE Journal on*, 20(8):1489–1499, 2002.

[5] EuQoS web page, avail. online: http://www.euqos.eu, 2008.

[6] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[7] D. Milic, M. Brogle, and T. Braun. Video broadcasting using overlay multicast. In *ISM '05: Proceedings of the Seventh IEEE International Symposium on Multimedia*, pages 515–522, Washington, DC, USA, 2005. IEEE Computer Society.

[8] OMNET++, avail. online: http://www.omnetpp.org, 2007.

[9] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environ ment. In *SPAA '97: Proceedings of the ninth annual ACM symposium on Paralle l algorithms and architectures*, pages 311–320, New York, USA, 1997. ACM.

[10] S. Ratnasamy, M. Handley, R. M. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*, pages 14–29, London, UK, 2001. Springer-Verlag.

[11] A. I. T. Rowstron and P. 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.

[12] D. A. Tran, K. A. Hua, and T. Do. Zigzag: an efficient peer-to-peer scheme for media streaming. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, volume 2, pages 1283–1292, 2003.

[13] TUN/TAP, avail. online: http://vtun.sourceforge.net/, 2007.

[14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and. Technical report, 2001.

[15] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 11–20, New York, USA, 2001. ACM.