*Published as: E. Frécon and M. Stenius, "DIVE: A Scaleable network architecture for distributed virtual environments", Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments), Vol. 5, No. 3, Sept. 1998, pp. 91-100.*



---

# DIVE: A scaleable network architecture for distributed virtual environments

Emmanuel Frécon - *emmanuel@sics.se*
Mårten Stenius - *mst@sics.se*

*Swedish Institute of Computer Science*
*Box 1263*
*SE-164 28 Kista*
*Sweden*

## Abstract

*We introduce the network software architecture of the Distributed Interactive Virtual Environment (DIVE). The platform is designed to scale to a large number of simultaneous participants, while ensuring maximum interaction at each site. Scaleability is achieved by making extensive use of multicast techniques and by partitioning the virtual universe into smaller regions. We also present an application-level backbone that can connect together islands of multicast aware networks.*

## 1. Introduction

Our goal is to build a framework for controlling the large data flows that are generated by collaborative virtual environments. Indeed, while creators of multi-user virtual environments often wish them to be spatially extended, when not infinite, to contain many detailed objects and to allow many simultaneous participants to interact with each other and with the environment, this would not be possible without the use of specific techniques.

The remainder of this paper is structured as follows. In the next section, we describe the conceptual abstraction through which DIVE applications transparently communicate with each other, namely virtual worlds. We then introduce the design choices that allow DIVE to scale to a large number of users interacting in real time. Following this, we present the communication architecture of the platform and the different techniques with which applications communicate event and streaming data. Then, we describe two key applications that allow DIVE to run on the Internet and adapt itself to its heterogeneity. Finally, we compare the DIVE architecture to other approaches.

In the first sections, we describe the DIVE philosophy using an argumentation roughly parallel to the architectural discussion developed in [Waters97] to describe the *Spline* system. Although DIVE and Spline have evolved separately, the platforms share many common goals and design choices. We discuss some differences between the two systems in more detail in section 6.2.

## 2. The World as a Common Interaction Medium

DIVE [Hagsand96] provides an architecture for implementing multi-user interactive virtual environments. The architecture focuses on software and networking solutions that enable high interaction at each participating site, i.e. interaction results are immediately shown at the interacting sites but slightly postponed at remote sites.

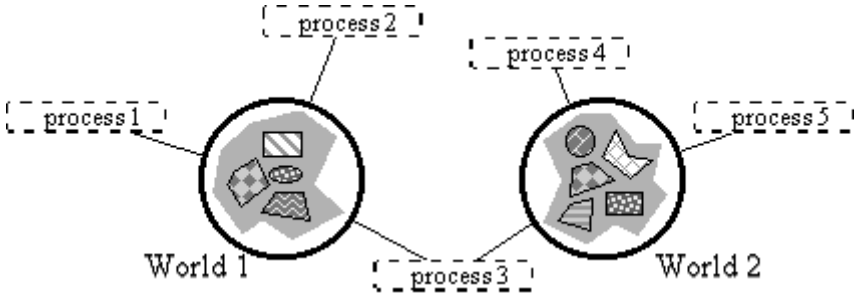


*Figure 1: In DIVE, separate processes interface and interact through one or more distributed world databases.*

A central feature in the programming architecture of DIVE is the shared, distributed world database. All user and application interactions take place through this common medium. This is illustraded in Figure 1, which shows five applications interacting through two different worlds. Application processes 1, 2, and 3 interact with each other through World 1. Applications 3, 4, and 5 interact with each other through World 2. Note that application 3 is interacting with both worlds.

DIVE applications operate solely on the world abstraction and do not communicate directly with each other. This technique allows a clean separation between application and network interfaces. Thus, programming will not differ when writing single-user applications or multi-user applications running over the Internet. This model has proven to be successful while DIVE has changed its inter-process communication package three times and applications did not require any redesign, only minor code tweaking.

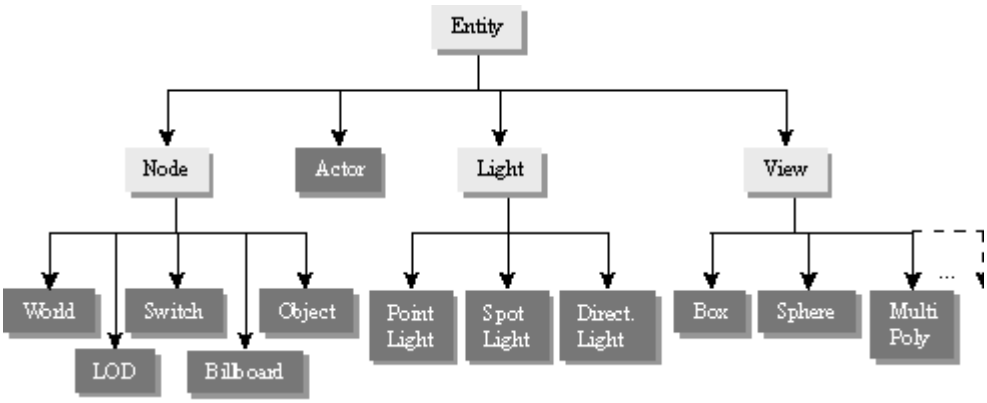


*Figure 2: The simplified DIVE entity class hierarchy.*

A DIVE world is a hierarchical database of what is called *entities*. DIVE entities can be compared to objects in object-oriented approaches, although DIVE is written in plain ANSI C. Entities are organised in a class hierarchy, depicted in Figure 2. The database is hierarchical to ease ordering of information and rendering. In addition to graphical information, DIVE entities can contain user-defined data and autonomous behaviour descriptions. We believe that the three-dimensional rendering module should not dictate a structure to the content of the database. By making this distinction the system becomes more flexible, and can more easily be adapted to a large variety of rendering techniques.

Entity persistency is ensured as long as one application interacts with a world. When the last application "dies", the world dies and entities will stop living. The next time an application connects to this world, it will reload its state from some initial description files, which are located by a file URL, and thus may be located either on local storage, or remote servers on an internet. The current DIVE version handle persistency by running an impassive process that periodically saves the world state to a file, in order to achieve failure-less restarts. This technique has to be refined – many issues are still to be investigated in depth, such as how to delegate responsibility for persistence of different parts of a world, and how to handle object ownership in a general way.

## 3. Partial, Active Database Replication

The DIVE architecture is based on active replication of (parts of) the database, so that a copy resides at each application process, as depicted in Figure 3. This model allows applications to access the world database directly from memory, which provides low-latency user interaction. In the following sections, we will call an application process running at a specific host a *peer*.

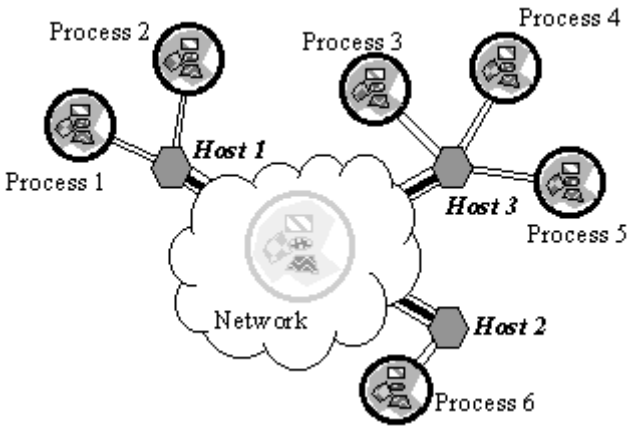


*Figure 3: The DIVE run-time architecture, simplified here to six processes interacting with each others in a single common world. Each process holds its own copy of the relevant objects, but the whole world can be seen as residing "on the network".*

Typically, entity additions, removals and modifications are done on the local copy first, then distributed to all connected peers through a network message and applied to the local copy at each receiving peer. By this we mean that the replication of the database is *active*. Conceptually, programmers can think of a "global" central database residing somewhere on the network, as in Figure 1, but the database is indeed replicated at each process, as in Figure 3.

Traditional distributed database systems often enforce that all processes agree on the information state before committing any updates. Such operations are difficult to combine with scaleable real-time interaction. DIVE, on the other hand, focuses on highly interactive applications, and uses *partial* replication to remedy some of these problems:

- DIVE tolerates that world copies differ slightly and implements services that ensure their equality over time. Dead-reckoning techniques and run-time object update mechanisms are used to achieve this.

- DIVE divides the world into sub-hierarchies that are only replicated and used in-between the small number of applications that have expressed an interest in them.

Modifications to the database are applied locally first, then packed, transmitted and applied on all receiving copies. Thus, on the receiving side, world events are captured and transcripted into the database some time after they occurred. The length of this time is highly related to the network carrier separating the processes. Round-trip estimates measured in DIVE experiments over the Internet range from 25 milliseconds within Sweden, around 200 ms from Swedish to American sites and 800 ms to Japan, along with high packet loss rates.

Despite this network latency, DIVE does not introduce differences between peers that are excessively large. The distribution principle fits with our experience of virtual environments: typically, entities will be modified in a bursting way and then stabilise into a common "inertia", i.e. a common state that lasts. Differences introduced by latency and lost network packets are "healed" over time by periodic synchronising using sequence numbers to keep track of entity versions, followed by possible update requests. As a result, with time all connected peers will achieve the same "inertial" state for these entities.

To remedy problems with network congestion and traffic overhead, and to offer possibility for hundreds of participants to share a common place, DIVE provides a mechanism for dividing the world into sub-hierarchies that are only replicated and used in-between the small number of applications that are actually interested in them. Each sub-hierarchy may be associated with a multicast communication channel, called a *light-weight group*. As a result, processes that are not interested in these sub-hierarchies can simply ignore this branch of the entity tree and cut down network traffic. (The top-most entity, i.e. the world, is itself always associated with a multicast channel, and every world member process must listen to it). Light-weight groups are discussed in more detail in section 4.3.

### 4. DIVE Communication Architecture

## *4.1 Peer-to-peer Communication*

DIVE focuses on peer-to-peer multicast communication as opposed to a "classic" client-server model. A typical client-server architecture would require DIVE processes to communicate with a server each time an entity of the database is to be modified. Thus, such an architecture would have negative implications on interaction time and would introduce lags.

There are two types of messages sent by DIVE, they correspond to the different kinds of data sent during a session:

- DIVE assumes that all entities are likely to be modified often. Thus, database modifications are sent using a reliable multicast protocol detailed in the next section. Multicasting is used to minimise the number of message duplications between the sender and the receivers. *Reliable* multicast is necessary to ensure the postponed equality of all world copies.

- Continuous data streams (i.e. audio and video) are sent using unreliable multicast. Here, the emphasis is not on maintaining consistency between different replicas but on achieving sound or visual continuity from a series of distinct messages.

DIVE messages comply with the RTP v2.0 (Real-time Transport Protocol) recommendation [Schulzrinne96] so that DIVE sessions can coexist with other MBone applications on the same multicast channel.

## *4.2 Networking and multicast protocol*

For non stream-based data communication, DIVE pursues an idea originally in SRM (Scaleable Reliable Multicast) [Floyd95] to reduce the amount of message passing and thereby minimise network load and increase scaleability. The method uses multicasting heavily, makes communication entity-based, and bases reliability on a negative acknowledgement request/response scheme.

In our approach, communication objects are equivalent to DIVE entities. A sending protocol peer does not need to store messages until their arrival has been confirmed by all recipients, as in TCP for example. Instead, it may request the latest object replica from its local application by a callback.

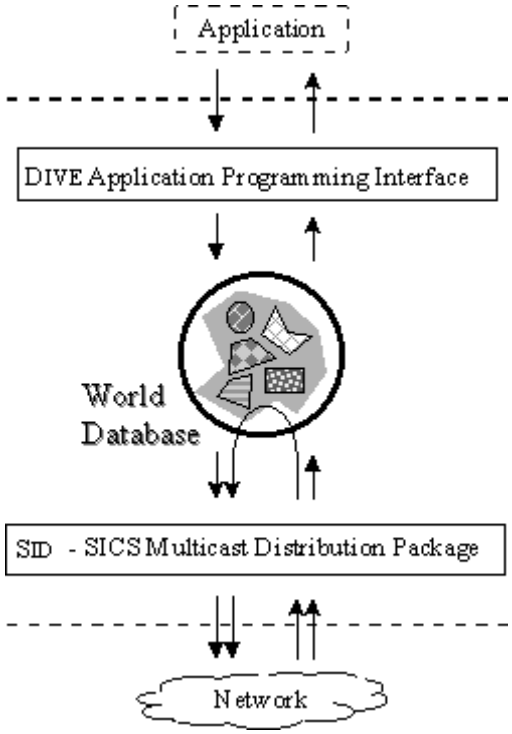


*Figure 4: A simplified view of the communication flow in a DIVE process. Note the active communication between Sid and the world database, made necessary by SRM callback mechanisms.*

If a peer detects a missing update message or just requests an object, it simply requests the object on the associated multicast address. By round-trip-time estimation and a timeout algorithm, the closest peer with the latest version of the object responds, also by multicast to inhibit other similar replies. In this way, the network is not flooded by replies. In the optimal case, only the closest peer replies. The DIVE process architecture and the callback mechanism from the inter-process communication layer into the world database are depicted in Figure 4.

A reliable service needs support from the application in the form of callbacks. Therefore, our transport service (called Sid) is implemented in user space and is an integral part of the DIVE software. However, the protocol has been used for other purposes, such as the JetFile file system for gigabit networks [Pink95].

## *4.3 Light-Weight Groups*

As mentioned before, DIVE uses multicast communication at two different levels of the world hierarchy:

- The top-most entity of the hierarchy is associated with a multicast group that will be used by default for all data communication within the world, unless the following case is satisfied.
- Any entity of the hierarchy can be associated with a different multicast group. When a modification message concerning an entity is to be sent, the algorithm will climb the hierarchy, starting from the entity. As soon as a multicast group is found, it will be used as a communication medium. If no group is found during the ascension, DIVE will use the default world group associated to the top-most entity.

Light-weight groups can be associated orthogonally to the entity tree, i.e. several branches can be associated with the same group. The light-weight group mechanism can be interfaced and controlled from a high level of abstraction using the DIVE/TCL scripts that can be associated to each DIVE entity. This gives us a more flexible tool that makes it

possible to experiment with a variety of application-dependent distribution schemes without requiring a hard-coding of the semantics into the platform. For example, this model is flexible enough to serve as a basis for experiments with different platform techniques such as aura intersection [Fahlén93] [Hagsand97] or subjective views [Snowdon95].

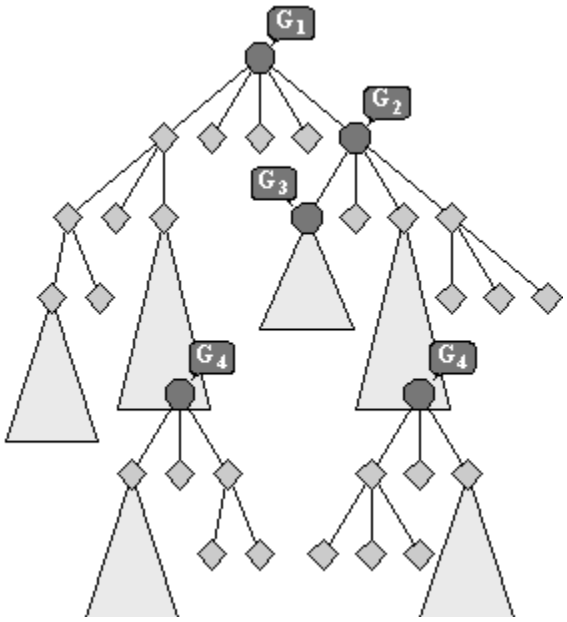


*Figure 5: An example of a world hierarchy, partitioned using light-weight groups.*

Figure 5 depicts an example of a world hierarchy containing light-weight groups. Entities associated to a mulitcast group are represented as an octagon marked with the name of the group. Other entities are represented by diamonds and entire sub-trees are simplified as triangles. $G_1$ is the world default group, while $G_2$ to $G_4$ are light-weight groups. Note that $G_4$ is used by two different branches of the hierarchy.

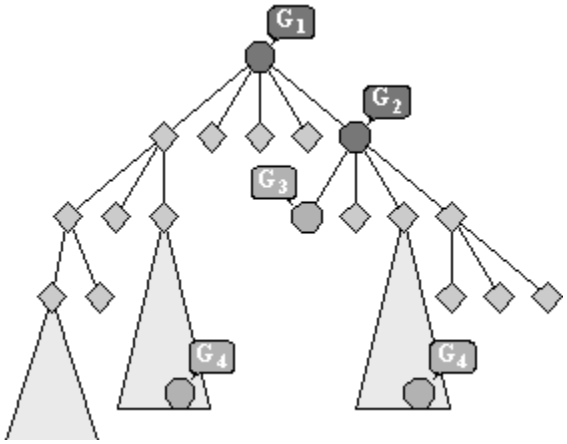


*Figure 6: A plausible view of the entire database of Figure 5 as seen from a peer.*

It is up to the process itself, and thus to the application, to decide whether to join a group or not. The decision can be made through autonomous scripts associated to entities or by other running applications. Figure 6 depicts an example of which entities of Figure 5 are actually replicated at a given peer. The peer has joined $G_1$ and $G_2$, but not $G_3$ and $G_4$. The light-weight groups that are not joined are marked with a lighter background.

## *4.4 Data Streaming*

DIVE sends continuous streams of data (i.e. audio and video) using unreliable multicast. Here, the emphasis is not on maintaining consistency between different replicas but on achieving sound or visual continuity from a series of distinct network messages. Reliability can not be a key issue since packets resending would only introduce lags in the continuous media and have negative implication on interaction.

DIVE associates separate audio and video multicast groups to each world. These groups are used for live communication. The current implementation does not associate separate audio and video to entities that carry a light-weight group reference. This implementation hampers scaleability for obvious reasons and we are planning to remedy this problem.

For audio and video transmission, DIVE inherits some techniques from the popular MBone tools VAT and VIC [Kumar95]. In particular, several encoding and compression methods are offered, in order to achieve a satisfactory balance between audio/video quality and network load. Finally, to improve immersion, point audio sources (i.e. object and avatar sounds) are mixed and spatialised to render a "soundscape" that corresponds to the three-dimensional structure of the world database [Adler96].

### 5. DIVE Run-Time Architecture

## *5.1 The DIVE Name Server*

The key application that allows DIVE processes to enter a world is the *diveserver*. The diveserver is a name server that exchanges a world name into a "ticket" that will let in an application willing to connect to a world. This "ticket" is simply composed of a multicast channel on which all further world communication will take place, except for specifically marked sub-hierarchies, i.e. light-weight groups. The workload of the diveserver is thus proportional to the number of processes that are willing to *enter* a world at a given time point, *not* to the total number of *connected* users during the session. Thus, while the diveserver is the necessary centre point with which any DIVE process needs to communicate before being able to enter a world, it scales easily to hundreds of simultaneous participants since it only needs to deal with initial connection requests.

Initial communication with the diveserver can be established either on a fixed multicast channel, or using a traditional point-to-point client-server request. Several diveservers can coexist on the Internet, but currently, applications that obtain their tickets from different diveservers will not be able to share the same world since they are likely to receive different global world communication channels from the different name servers.

Once a process has received its "ticket", it will not communicate with the diveserver anymore. Instead, it will ask for the current world state by sending a request to the world multicast channel. If the process is the first one to connect, it will read the initial world status from several Internet locations. Otherwise, it will receive an incremental state from already running processes, using regular point-to-point communication or multicast. As this state can represent consequent amount of data, its total size is reduced using ZLib compression [Deutsch96]. DIVE uses a round-trip-time algorithm to find the nearest processes likely to answer.

## *5.2 The DIVE Proxy Server*

For sessions involving peers located in different local networks, DIVE has long relied solely on the existence of the MBone [Kumar95] — the IP Multicast backbone, a structure for interactive multimedia communication over the Internet. From the DIVE point of view, the MBone presents some weaknesses:

- Despite its wide acceptance within the research community, the MBone spreads slowly between sub-networks. Thus, running a world-wide DIVE session can require the intervention of several intermediate system staffs.

- Even if most of the operating systems now support multicast in their standard installation — sometimes as an option — some older versions do not or can not be upgraded.

- At the corporate level, multicasting is still weak. As long as the corporate local network is composed of relatively new operating systems, it has an inherent support for multicast. However, MBone connection, i.e. multicast connection with the outer world, is often missing.

- IP multicasting is not fully supported on different carriers such as ATM or ISDN lines.

Therefore, independence from the MBone was recently gained by developing the DIVEBONE, an application-level backbone which can interconnect sub-islands with multicast connectivity and/or single local networks. The key application to the DIVEBONE is the DIVE proxy server. This application provides two different services on DIVE aware networks:

- The proxy server allows multicast unaware processes to join regular multicast sessions by acting as a message replicator for all connected DIVE applications. This is depicted in Figure 7 and detailed below.

- The proxy server allows MBone unaware networks to support DIVE multicast sessions by acting as a message replicator for all connected proxy servers. This is depicted in Figure 8 and detailed below.

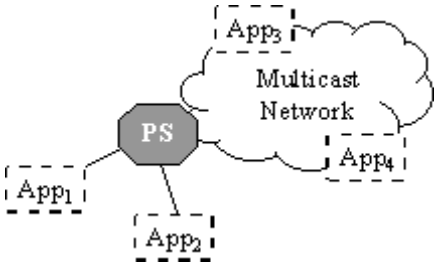


*Figure 7: The DIVE proxy server as a simple message replicator.*

The proxy server can act as a simple message replicator for individual clients. It catches all DIVE multicast messages sent by applications and replicates the messages to all connected clients. Similarly, each time a client sends a message, the proxy server will replicate the message to all other connected clients and to the multicast group. As an example, in Figure 7, PS is a proxy server that will forward all data from $App_1$ to $App_2$ and the multicast network, and thus these messages will be caught and interpreted by $App_2$, $App_3$, and $App_4$. Similarly, all messages sent by $App_2$ will reach $App_1$, $App_3$ and $App_4$. PS will also forward messages coming from the multicast channel, i.e. from $App_3$ or $App_4$, to $App_1$ and $App_2$.

For simplicity, in the previous example we have considered that all the running DIVE applications had joined the same session, and thus were communicating through the same world database. However, the DIVE proxy server is able to perform software, high-level multiplexing in-between different groups.

The current version of the DIVE proxy server is not aware of the worlds within which the different applications interact. It simply acts as a message dispatcher at the multicast level. However, a straightforward evolution is to build intelligence into the proxy server and use it to compress and pre-format data before sending it to its direct clients, allowing them to run on low-bandwidth networks such as modem lines. Examples of pre-formatting include audio spatialisation and mixing. Another evolution is to automatically duplicate the proxy servers in order to cope with the additional computational and network load introduced by new clients. Computational load is likely to increase if compression and pre-formatting are introduced.

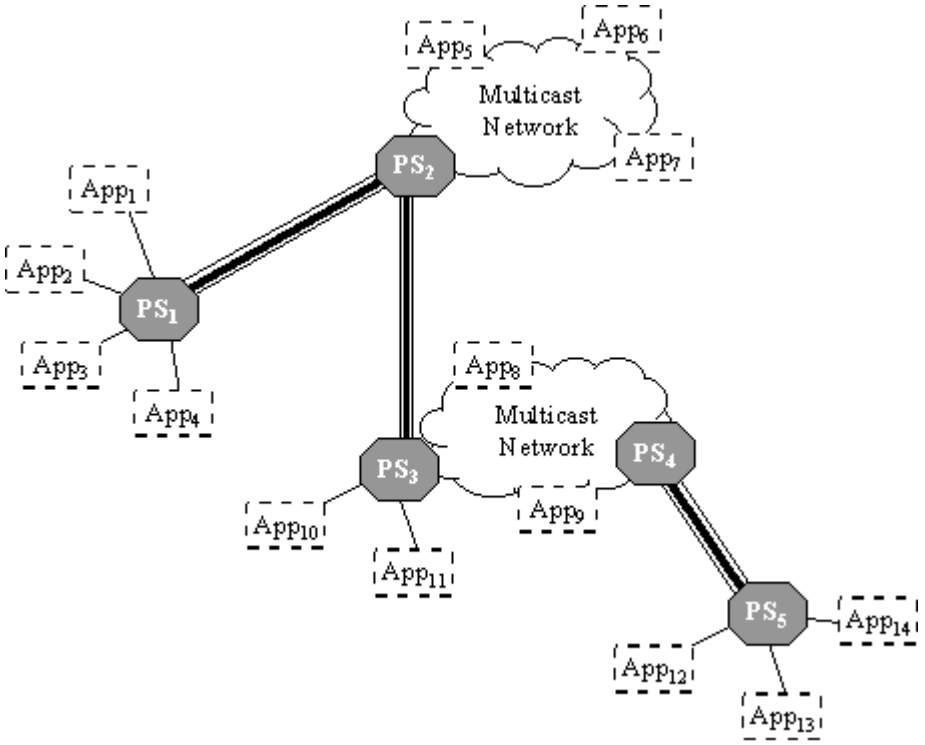


*Figure 8: The DIVE proxy server as an application-level multicast tunneler.*

The proxy server can also act as a multicast tunnelling application: Additionally to serve directly connected DIVE applications, proxy servers can also connect to each other in order to bind together MBone unaware multicast networks. Figure 8 depicts a complex example of five different proxy servers, named $PS_1$ to $PS_5$, that allow communication between fourteen different applications, named $App_1$ to $App_{14}$.

For simplicity, in this example we consider that all the running DIVE applications have joined the same session, and thus are communicating through the same world database. But, again, proxy servers are able to perform software, high-level multiplexing in-between different groups and serve different applications on different groups. Tunnelling proxy servers with each others constructs the DIVEBONE.

Because of the complexity of setting up a DIVEBONE architecture, proxy servers are guarded against cycles. In short, packets entering the DIVEBONE are marked with the identifier of the proxy server that introduced them and proxy servers drop packets that are received from two different sources. The DIVEBONE uses principles that are similar to the first versions of the multicast routing daemon, i.e. DVMRP [Pusateri97].

A straightforward evolution of the DIVEBONE is again to introduce compression and pre-formatting in-between proxy servers. Another interesting application of the DIVEBONE is application-level network analysis. The current version of the proxy servers can log network traffic and a companion application shows, in real-time, some graphical statistics. The most interesting statistic gather separate logs for each message type sent by DIVE applications. Using these statistics, it has been possible to reduce the packet size for the most common operations, and thus to increase scaleability.

## 6. Related Platforms

The systems that are most related to DIVE are those that have been experimenting around the same ideas, such as loose consistency, no central servers and world sub-partitioning.

## 6.1 The Ancestors — SIMNET and DIS

In order to achieve scaleability, i.e. hundreds of simultaneous participants, DIVE takes an approach that was pioneered by SIMNet [Calvin93] and DIS [Dis93]. The key to this approach is that a distributed scaleable system can only achieve real-time performance if the view of the common data is not fully consistent throughout all the peers that participate to the simulation. DIVE share some commonality with SIMNet and DIS:

- As described above, DIVE makes wide use of peer-to-peer communication.
- DIVE does not rely on any central service (except the diveserver), for improved scaleability and fault tolerance.
- DIVE uses dead-reckoning mechanisms to reduce the number of messages generated when objects move.

However, DIVE goes beyond DIS on some key aspects:

- DIVE is a complete platform for implementing interactive multi-user virtual environments. DIVE is not only a communication standard.
- DIVE supports both audio and video communication. We believe that spatialised audio, and in particular live communication, is a key to the success of virtual environments.
- Introducing new objects at one site does not require that the other sites can access an external description of the objects, typically a file containing three-dimensional data. When a peer introduces new entities in the world, these are automatically transmitted to all connected sites, using DIVE specific mechanisms.
- Peers do not necessarily own objects. As a result, a peer will be able to connect, introduce an autonomous object and leave. The autonomous object will continue living, even after the peer has left.
- DIVE divides the world into sub-regions that are associated with light-weight communication groups. While NPSNet (Naval Postgraduate School Net) [Macedonia95] also supports sub-partitioning, it is less flexible since the sub-regions have a predefined shape, i.e. tiles of hexagons.
- DIVE network messages do not contain whole objects, but rather self contained modifications to be applied to entities.
- DIVE does not require that peers send keep-alive messages. Instead, we have based our approach on SRM and are able to detect missing data packets on reception.

Much research on networking and protocols is being done in the context of DIS and DIS-like systems. Of particular relevance to the DIVEBONE architecture is the currently planned *CBone (Cyberspace Backbone)* [Brutzman96], which also aims to serve as a application-driven replacement for the MBone, providing new services such as bandwidth reservation and dedicated multicasting. Furthermore, among current work on reliable multicasting is *LBRM (Log-Based Receiver-reliable Multicast)* [Holbrook95], which provides a way for detecting data loss on the receiving side, by employing "heartbeat" messages and distributed logging.

## 6.2 A Comparison: The Spline System

The Scaleable Platform for Large Interactive Network Environments (Spline) ([Barrus96] and [Waters97]) is one of the software platforms that resemble DIVE most. Similarly to DIVE, Spline uses peer-to-peer communication and a derivative of SRM. As DIVE, Spline has evolved from a pure multicast approach to a mixed client-server and multicast

approach, in order to cope with low-bandwidth networks. The role of the Spline session manager can be loosely compared to that of the diveserver, even though the role of the diveserver is not as interactive, but rather to be seen as a mostly passive name mapping service.

Spline divides the universe, which is called the *world model*, into sub-regions called *locales*, each associated with a multicast group, a technique that can be compared to the light-weight groups of DIVE. DIVE supports coarse-grained partitioning of the whole universe by introducing worlds and gateways between worlds. DIVE also supports fine-grained partitioning of the worlds by introducing light-weight group, a flexible application-dependent mechanism for implementing sub-regions. Spline, however, does not introduce this partitioning dichotomy, introduces locales as a complete concept for interrelating and subdividing worlds and regions.

Spline has abandoned Scheme - a scripting language - as its language for autonomous object description; instead, JAVA, a strongly structured language will be used. We believe that scripting languages are more appropriate, since they allow rapid and interactive prototyping. DIVE uses the Tool Command Language (TCL) [Ousterhout94] as its main behaviour language and have directly benefited from the release of version 8, which introduced a byte-code compiler and offered an improved execution speed. DIVE supports JAVA through a prototype external JAVA-to-TCL interface, which hides TCL under a strongly typed class hierarchy.

## *6.3 Community Place*

*Community Place* [Lea97a] is a system developed at Sony, that partially stems from a research collaboration with SICS [Lea97b]. The system uses a layered communications architecture where client-server communication is used on low-bandwidth networks, i.e. on modem lines to the actual users, and peer-to-peer communication is used for long-distance communication between servers. The browser is tuned for low-cost PC hardware, to promote a wide spread of the platform, uses VRML 2 for world descriptions, and features server-based Java as a behaviour language.

## *6.4 Massive-2*

*MASSIVE-2* [Greenhalgh96] [Greenhalgh97] is developed at the University of Nottingham. It is based on the use of *third-party objects* [Benford97b], which act as mediators of awareness and relationship between other objects in the environment. The third-party objects are used as a general mechanism for filtering, structuring and partitioning of the shared environment, with IP Multicasting as a basis for communication. MASSIVE-2 addresses issues such as nested world structuring, dynamic view abstraction and aggregation, and the management of common foci for several participants.

## *6.5 Other Recent Work*

Recently, much practical work on shared virtual environments on the Internet has focused on the development of the Virtual Reality Modelling Language (VRML) [Vrml97]. Most of these systems support a static background scene and loose sharing of avatars. Furthermore, most platforms rely on simple client-server communication, often together with a single centralised server. As we have pointed out before, interactive client-server based systems suffer from higher latency and from a difficulty to scale up. Finally, these systems usually support only distribution of avatar motion, at a filtered low-frequency update. Therefore they are able to claim high numbers of simultaneous participants. DIVE, on the contrary, focuses on an architecture that supports unfiltered free motion of any entity, including avatars and other reacting autonomous objects, in order to achieve high-quality interaction and communication between participants.

### 7. Conclusion

In this paper, focus has been put on how to support a system that is scaleable both with respect to the number of users, and to the network distance between different users on heterogeneous networks. In particular, two such techniques stand out:

*The DIVE proxy server and the DIVEBONE* provides tools for an application-level virtual Multicast backbone, that can be used in heterogeneous settings where connectivity in other ways are limited, and to create distributed, but separate, application-specific DIVE networks. Furthermore, DIVE becomes a hybrid system that can both be viewed as a client-server and a peer-to-peer system, with increased flexibility for varying demands.

*Light-weight groups* are a database-level mechanism that allows a partitioning of distributed environments into subsets that can be requested or disregarded based on custom semantics. This technique can be used as a general high-level, application-dependent schemes for experimenting with issues such as optimizing data transfers, database sizes, view culling, object and world abstractions, and so on.

Achieving scaleable virtual environments requires an adequate architecture such as the one described in this document. However, we believe that application-level techniques are also required to reduce the number of network messages sent and received by peers that are connected to a common environment. The common theme of the techniques that we are experimenting with is that they intend to replace a group of low-level unary operations by semantically rich operations that only require a few network messages for their transmission. We are currently planning to implement or implementing the following:

- Operation aggregation controlled from the application. This is particularly useful when realistically animating virtual humans: all limbs rotations can be grouped together into a single operation.
- Adding an animation language that is triggered on event reception. Animations are distributed to all peers and thus can be executed locally and simultaneously at all peers.
- Interfacing a scripting language. Scripts are distributed together with the database and can, in some circumstances, be executed locally and simultaneously at all peers. While animations are tuned for visual authoring tools, scripts can be used to add intelligence to the animations and glue together different animations.

The DIVE system has developed from a lab tool to an evolving, general-purpose platform for networked, shared virtual environments. While this paper has focused on the database and network levels, the system is used for experimentation with application, user interfaces and interaction in a wide variety of settings. DIVE is an open research platform, binaries are available for downloading, for non-commercial use, at http://www.sics.se/dive/.

## 8. Acknowledgements

We would like to thank three key persons without whom DIVE would never have existed: Olof Hagsand has been the primary architect of the DIVE platform for a long time and many of the principles and design choices described in this article are direct results of his research. Olov Ståhl has been working on DIVE almost since the project started and his knowledge and understanding of all the internal mechanisms are extremely valuable. And without the visions and strong will of Lennart Fahlén, the platform would never have become the general testbed for shared virtual spaces it is today.

## References

[Adler96] D. Adler, "*Virtual Audio - Three-Dimensional Audio in Virtual Environments*", SICS Internal Report, ftp://ftp.sics.se/pub/SICS-reports/Reports/SICS-T--96-03--SE.ps.Z, 1996, ISRN: SICS-T--96/03-SE.

[Barrus96] J. Barrus, R. Waters, D. Anderson, "Locales: Supporting Large Multiuser Virtual Environments", IEEE Computer Graphics & Applications, November 1996, pp 50-57.

[Benford97a] S. Benford, C. Greenhalgh and D. Lloyd "*Crowded Collaborative Virtual Environments*", Conference on Human Factors in Computer Systems (CHI'97), March 1997, Atlanta, Georgia.

[Benford97b] S. Benford and C. Greenhalgh, "*Introducing Third Party Objects into the Spatial Model of Interaction*", European Conference on Computer Supported Cooperative Work, Lancaster, UK, September, 1997, pp. 189-204.

[Brutzman96] D. Brutzman, M. Zyda, and M. Macedonia, *"Cyberspace Backbone (CBone) Design Rationale"*, 15th DIS Workshop on Standards for the Interoperability of Distributed Simulations, Institute for Simulation and Training, Orlando Florida, September 16-20 1996

[Calvin93] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller and D. Owen, "*The SIMNet virtual world architecture*", Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS), 1993, IEEE Computer Society Press, Los Alamitos, CA, pp. 450-455.

[Deutsch96] P. Deutsch, J-L. Gailly, "*ZLIB Compressed Data Format Specification version 3.3*", RFC 1950, May 1996.

[Dis93] "*Standard for information technology, protocols for distributed interactive simulation*", DIS-ANSI/IEEE Standard 1278-1993, American National Standards Institute, 1993.

[Fahlén93] L. E. Fahlén, O. Ståhl, C .G. Brown, C. Carlsson, "*A Space Based Model for User Interaction in Shared Synthetic Environments*", Proceedings of INTERCHI'93, April 24-29 1993, Addison-Wesley, pp. 31-37.

[Floyd95] S. Floyd et al., "*Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing*" Proceedings of SIGCOMM 95, September 1995, ACM Press, New York, pp. 242-256.

[Greenhalgh96] C. Greenhalgh, "*Dynamic, embodied multicast groups in MASSIVE-2*", Technical report NOTTCS-TR-96-8, Department of Computer Science, The University of Nottingham, UK, 1996

[Greenhalgh97] C. Greenhalgh, "*Analysing movement and world transitions in virtual reality tele-conferencing*", Proceedings of ECSCW'97, Lancaster, UK, September 1997.

[Hagsand96] O. Hagsand, "*Interactive Multi-User VEs in the DIVE System*", IEEE Multimedia Magazine, Vol. 3, No. 1, 1996.

[Hagsand97] O. Hagsand, R. Lea, M. Stenius, "*Using spatial techniques to decrease message passing in a distributed VE*", VRML'97, Monterey, CA, February 1997.

[Holbrook95] H. W. Holbrook, S. K. Singhal, D. R. Cheriton, *"Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation"*, Computer Communication Review, ACM SIGCOMM, vol. 25, no. 4, October 1995

[Kumar95] Kumar, Vinay, "*MBone: Interactive Multimedia On The Internet*", Macmillan Publishing, November 1995, ISBN: 1-56205-397-3.

[Lea97a] R. Lea, Y. Honda, K. Matsuda, S. Matsuda,"*Community Place: Architecture and Performance*": VRML'97 Monterey, CA, February 1997.

[Lea97b] R. Lea, Y. Honda, K. Matsuda, O. Hagsand, M. Stenius, "*Issues in the design of a scaleable shared virtual environment for the Internet*", HICSS-30, Hawaii, January 1997.

[Macedonia95] M. Macedonia., M. Zyda, D. Pratt, D. Brutzman and P. Barham, "*Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments*", IEEE Computer Graphics & Applications, September 1995, pp. 38-45.

[Ousterhout94] J. Ousterhout, "*Tcl and the Tk toolkit*", Addison-Wesley Publishing Co., 1994, ISBN: 0-201-63337-X.

[Pusateri97] T. Pusateri, "*Distance Vector Multicast Routing Protocol*", IETF - Work in Progress, draft-ietf-idmr-dvmrp-v3-05, October 1997.

[Pink95] S. Pink, A. Saulsbury, and O. Hagsand, "*OS 6 — A Distributed Operating System for the Next Generation of Computer Networks*," Proceedings of the IEEE International Workshop on Object-Orientation in Operating Systems (IWOOOS 95), August 1995, IEEE Press, Piscataway, N.J., pp. 48-51.

[Schulzrinne96] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "*RTP: A Transport Protocol for Real-Time Applications*", Internet Draft Standard, RFC 1889, January 1996.

[Snowdon95] D. Snowdon, C. Greenhalgh, and S. Benford, *"What you see is not what I see: Subjectivity in virtual environments"*, Framework for Immersive Virtual Environments (FIVE'95), London, UK, December 18-19 1995

[Vrml97] "*The Virtual Reality Modelling Language*", ISO/IEC Draft for International Standard 14772-1, http://www.vrml.org/Specifications/VRML97/DIS/, April 1997.

[Waters97] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns and W. Yerazunis, "*Diamond Park and Spline: Social Virtual Reality with 3D Animation, Spoken Interaction and Runtime Extendability*", Presence, Vol. 6, No. 4, August 1997, pp. 461-481.