

# An Open Protocol for Wide-area Multi-user X3D

**Jay C. Weber**

Media Machines, Inc.  
San Francisco, California, USA  
jweber@mediamachines.com

**Tony Parisi**

Media Machines, Inc.  
San Francisco, California, USA  
tparisi@mediamachines.com

## ABSTRACT

This paper describes work to create an open protocol for wide-area multi-user X3D, incorporating many aspects of prior academic, experimental, and proprietary systems, but emphasizing simplicity and practicality for use among heterogeneous Internet user agents. In the Internet tradition, it is documented as a protocol (rather than a framework), and backed by freely-available reference implementations. The hope is that this protocol is useful to those working on new X3D networking nodes as well as to those building multi-user world systems.

**CR Categories:** C.2.2 [Network Protocols]: Applications; C.2.4 [Distributed Systems]: Distributed Applications; I.6.8 [Types of Simulation]: Distributed, Gaming

**Keywords:** multi-user 3D, X3D, Web3D, virtual reality, online game programming.

## 1. INTRODUCTION

This paper describes a communication protocol in support of X3D scenes synchronized across potentially-many users connected by wide-area networks (the Internet, in particular). Our goal was to incorporate architectures and features from the wealth of prior work [IEEE 1993, Singhal and Zyda 1999, White 1998, Kimball 2006, Blaxxun 2006], optimize for synchronization of X3D scene-graphs, boil it down to the simplest platform that is practical over the Internet, and make it open.

The result is a Simple Wide-Area Multi-user Protocol (SWAMP) and open-source reference implementations of clients and servers. While we believe that SWAMP is a new combination of good protocol design elements, the point of this paper is not the novelty of the individual design elements (most are well-known), but to consolidate a practical and flexible protocol, open to all, that enables a new generation of robust experiments and applications in multi-user X3D.

## 2. PROTOCOL DESIGN

### 2.1 Entity-state Model

A multi-user protocol conveys state-change information among user agents (the software users use to see the shared state). Thus the fundamental question in the design of such a protocol is: what are the entities that have dynamic states. All multi-user protocols start with an entity-state model.

Most multi-user protocols define their own model, and applications of the protocol must map the protocol's entity-state model to the model(s) used by state renderers and reasoners. Since we target X3D clients specifically, SWAMP directly adopts X3D's scene-graph model. That is, SWAMP describes changes to fields of scene-graph nodes. See [Web3D 2003] for the details of X3D's model, but if one is more familiar with the structure of XML, it is useful to think analogously of XML's tags/nodes and attributes and the Document Object Model (DOM) approach to manipulating them.

SWAMP gains several efficiencies by adopting X3D's entity-state model. First, the semantics and mechanics of the way X3D clients map between the protocol and renderer models is trivial, enabling efficient and correct implementations. Secondly, the protocol's ability to convey scene-graph changes is extremely general and very concise. Thirdly, infrastructure agents (including centralized-servers) can easily adopt an X3D scene-graph internal model, enabling re-use of geometric reasoning algorithms from existing clients (e.g., collisions, line-of-site, and proximity, when brokering multi-user interactions).

These efficiencies are an improvement over the DIS component of the X3D specification [Web3D 2003], based on influential but early work on distribution simulation. DIS [IEEE 1993] embedded the semantics of entities and their states in the protocol data units, leading to complex and inefficient messaging, and a semantics that is rather specific to military simulations. X3D's DIS component inherits both of those drawbacks, which is why we didn't use it as the literal basis of SWAMP.

Again, as opposed to DIS but also other more recent multi-user 3D protocols [White 1998], the SWAMP protocol defers the application-specific semantics of entities and their states to the client and server applications that use it. In so doing, SWAMP achieves a practical simplicity and a useful generality.

### 2.2 Message Encoding

Most Internet application-layer protocols, including those for email and web, favor human-readable plaintext encodings of protocol messages for reasons of simplicity and perspicuity. We would agree on this for SWAMP except that practice has shown bandwidth and latency to be extremely critical factors in the performance of multi-user 3D systems. For this reason we departed with Internet tradition and have made SWAMP a packed binary protocol, as described in Section 3.

### 2.3 Message Transport

Most Internet application-level protocols use the TCP stream-connection abstraction as their underlying network transport. TCP provides for guaranteed and order-preserved delivery of data, making for simple, predictable communication, and it is also the

most widely- and well-supported transport by programming environments, and infrastructure.

Despite TCP's advantages, much work on Networked Virtual Environments, in research [Singhal et al 1997] as well as online gaming products (e.g., Everquest), has focused on the use of UDP datagrams because of their relatively low overhead and latency. The most common application of such datagrams is for the communication of the rapidly-changing position of an object in the scene, where low-latency is a far more important consideration than guaranteed delivery (if one position update is lost then another will arrive soon) and order-preservation (if updates arrive out of order, just use the most recent).

SWAMP's approach to TCP vs. UDP transport is to support both. All communication starts with a TCP connection, given its relative ubiquity, but then that communication channel can be used to agree on the use of UDP datagrams as well. Thus the two transports would be used as appropriate for different kinds of messages, e.g., UDP for rapidly-changing state, and TCP for slowly/infrequently-changing state (where guaranteed delivery is important). Note that due to factors including firewalls, it may not be possible to use UDP datagrams in both directions, so the TCP connection could be used as a fallback for all messages (but with potentially degraded performance).

Some networked virtual environments have used IP/multicast of datagrams as a transport. When available, this has compelling advantages in bandwidth efficiency. IP/multicast can be made available on private networks including LANs, but at this time is not practical over the Internet at-large. SWAMP has basic support for IP/multicast, for compatibility if and when multicast is more ubiquitous, or for local/private network configurations.

## 2.4 Communication Architecture

The communication architecture of a networked virtual environment is the way that the various computational agents are connected and the roles they play. The most common architecture in practice is a *star* or *central server* architecture [Blaxxun 2006, White 1998] but it is worth noting that many LAN-based multi-user games, military systems, and academic systems use a peer-to-peer architecture, including some of the seminal systems [IEEE 1993]. Some systems use an architecture that is a hybrid of central-server and peer-to-peer.

SWAMP can be used as a communication protocol within any of these architectures. However, SWAMP is intended for use with wide-area and therefore heterogeneous networks, where a central-server architecture is usually the best choice. The main reasons for this have to do with bandwidth and security.

Regarding bandwidth, note that when an end-user causes a change of shared state (e.g., by moving her avatar), news of that change must be communicated to all other users that are viewing that state. In a pure peer-to-peer architecture, this entails sending messages to potentially all other users, thereby consuming commensurate bandwidth. Outgoing bandwidth is usually a scant resource for Internet end-users (especially those with asymmetric bandwidth, such as DSL). With central-server, each user sends but one message describing a state change, and the central-server, which will have access to much more bandwidth, replicates the message across the other users.

Regarding security, systems such as competitive games must contend with end-user fraud. Peer-to-peer architectures by definition do not have a central authority, whereas central servers can enforce good behavior and arbitrate disputes about shared state.

## 3. MESSAGE SYNTAX

### 3.1 Basic Syntax

The (BNF) syntax of SWAMP messages is shown in Table 1. All numbers are to be encoded into message data using standard Internet big-endian byte-order.

The datagram-message construct is used when sending (UDP) datagrams. Since datagrams are connectionless and therefore self-contained, the ticket is necessary to identify the source of the message. The ticket is potentially long so that it can be used not only to identify the source, but also authenticate, and it is variable-length so that it can accommodate various authentication tokens.

The sequence-number is used to detect when messages arrive out-of-order when using datagram transports like UDP; if said seq-number is less than the highest seq-number recently received from the same source-id, but not very much less (which would indicate roll-over), then it should be ignored. Values for "recent" and "very much less" can be tuned for specific systems.

When using a stream-, connection-based transport (TCP), the datagram wrapper is unnecessary and base-messages are used.

Table 1. SWAMP Message Syntax

datagram-message	::= ticket-len ticket seq-number base-message
ticket-len	::= 0..65535 (2-byte)
ticket	::= byte-string
seq-number	::= 0..65535 (2-byte)
base-message	::= control-message   content-message
control-message	::= control-opcode parameters
control-opcode	::= 0...127 (1-byte)
content-message	::= content-opcode parameters
content-opcode	::= 1...127 (1-byte)
parameters	::= byte-string

### 3.2 Content Messages

At this time, and in the spirit of the simple, X3D-based entity-state model of section 2.1, content messages describe changes to an X3D scene graph.

Parameters are marshaled into a byte-string in the following way. The byte-serializations of each parameter are concatenated in the order specified. Stringz parameters are zero-terminated byte-strings. Integer-typed parameters are serialized with big-endian

byte order. The type parameter is an enumeration of those types, and serialization of the value parameter depends on the type.

The `set` message specifies a new value for the field named `field` of the named scene-graph node `node`. The `type` of said field could be analyzed by looking at the scene graph, but is included to make message processing easier for clients and other agents (and one byte for the type is not much).

When a field, such as the translation of a Transform node, is frequently changing, repetition of the node name, field name, and type can accumulate into a significant amount of bandwidth. The `setf` message can be used to save these resources in the following way: when a `set` message has a non-zero value for `fieldId`, the recipient of this message should cache the node, field, and type for future processing of `setf` messages that contain the matching `fieldId`.

Use of the `setf` message is potentially very important to efficiency. If we assume that the average node and field name length is 10 characters, using a `setf` for repetitive field-value changes, each `setf` saves 23 bytes per message. For a typical SFVec3F value change, this means the `setf` version will consume less than half the packet-payload bandwidth of the `set` message. Plus, the SWAMP content messages are more efficient than the large (> 100-byte) DIS [IEEE 1993] messages, especially when entity-state parameters are not all varying at once.

Table 2. SWAMP Content Messages

Op code	Name	Parameters
1	set	Stringz nodeName; stringz fieldName; signed byte type; unsigned short fieldId; varying value;
2	setf	unsigned short fieldId; varying value;
3	add	Stringz parentName, childName, nodestr;
4	remove	Stringz parentName, childName;

The `add` and `remove` messages support changes to the structure of the scene graph. The `add` message requests that the sub-graph created by parsing the `nodestr` be added as the last child of the existing parent node. The `parentName` and `childName` names refer to DEF'd node names in the scene graph, or lacking that resolution, to nodes with the given `childName` from previous `add` messages. (If an agent's Scene Access Interface does not provide for dynamic definition of DEF names, it may need to maintain a table of child names and node pointers from previous `add` messages.)

This is a subset of the content messages to completely cover all operations on X3D scene-graphs, including addition and removal of ROUTEs, as embodied in the X3D Scene Access Interface (SAI). Our intention is to extend the content message set accordingly.

### 3.3 Control Messages

Control messages provide information about inter-agent communication rather than the shared content. SWAMP's control messages are summarized in Table 3.

The `hello` message is the first message sent over a given transport, in both directions. Its two parameters provide a byte-string `id` for the sender, and a byte-string `ticket` that the sender would like the recipient to use when identifying itself in the future (e.g., in a `datagram-message`).

Table 3. SWAMP Control Messages

Op code	Name	Parameters
0	hello	unsigned short id-len, ticket-len; byte-string id, ticket;
-1	comm	unsigned short protocol; utf8 host; unsigned short port
-2	ack	unsigned short protocol; utf8 host; unsigned short port

The `comm` message is used to request additional message transports beyond the initial TCP connection. The protocol parameter is one of an enumeration of possible transport mechanisms (1=UDP, 2=multicast), and the host and port parameterize the transport.

The `ack` message is used to confirm the efficacy of a particular transport. That is, just because an agent is listening on a particular transport, and requests its use via a `comm` message, doesn't guarantee a viable path between the agents. So the recipient of a `hello` message over an alternate transport is required to send an `ack` over the initial TCP transport to confirm receipt.

## 4. SCENE-ENTITY ABSTRACTIONS

The fact that SWAMP uses an entity model based on X3D (nodes and fields) does not mean that SWAMP messages are limited to describing only "low level" scene graph changes. For example, it may not be efficient to animate an avatar by sending SWAMP messages describing every joint transform to every applicable user. Instead, SWAMP agents should agree on useful scene-entity abstractions, and describe changes to their fields.

X3D not only provides abstractions inherent in how child nodes inherit from parent nodes, but its PROTO construct provides a clean way to abstract a scene sub-graph into a new node type. We expect that most systems that use SWAMP will do well to structure their scene-graphs using instantiated PROTOs, and use SWAMP to describe changes to PROTO fields, as is done in the successful [Blaxxun 2006] system for avatars and objects.

However, our SWAMP-based multi-user systems extend previous object and avatar PROTOs in an important way, by adding a `velocity` field. The use of such a field gives the multi-user agents a much more concise way of describing linear trajectories, so that agents can calculate changes in position over time instead of requiring a substantial stream of position change messages.

This is similar to the use of velocity PDU fields in [IEEE 1993], and as with those systems, communication latency can make position calculations quite complex, leading to a class of *Dead Reckoning* algorithms. Such complexities can pay off in smoother object movements and drastically-better bandwidth

utilization. We also see the value of adding an *acceleration* field, especially for vehicle motions.

## 5. INTEGRATION WITH THE X3D SCENE GRAPH

Configuration of network connections, and multi-user messaging logic, can happen at the meta-level above the scene-graph, as in past systems [White 1998, Blaxxun 2006]. That is, the scene-graph itself doesn't know that it is multi-user, it is merely a data structure for logic that handles connectivity and state changes. Alternately, network configuration and designation of dynamic state could be embodied by nodes in the scene.

SWAMP can be the underlying state-change messaging protocol in either approach. However, we are particularly interested in latter approaches because we think that standardized network nodes will improve interoperability, and encourage better support for multi-user features in authoring tools.

In particular, SWAMP integrates well with the X3D Network Component proposed by [THORNE 2006], which defines `EventStreamSensor` nodes that establish network connections over which to synchronize the values of a collection of fields. SWAMP is well-suited to be the event packet transmission format for the `EventStreamSensor` nodes in that network component. [THORNE 2006] includes a basic transmission format but only as a default.

## 6. IMPLEMENTATION

The SWAMP protocol and some compliant client and server agents have been implemented by the authors in an experimental system called "Hydra." This implementation is available as an open-source project at <http://www.mediamachines.com>. Also, we are in the process of integrating Hydra into our open-source Flux Player, which we expect to be available at the time of this paper's publication.

We have intentionally left out SWAMP functionality in support of multi-user chat and related presence information (buddy lists, online status). Some systems, including some of our own, have added messages types to handle such information, and certainly the display of such information can have an impact on the scene graph. However, we feel that the handling of such functionality is better-served by existing open protocols, XMPP (Jabber) in particular.

One reason for keeping SWAMP and chat functionality separate is that chat information is largely *not* replicated wholesale across multiple users, and so they are different synchronization problems. That is, friend lists and chat lines are not shared, they are specific to each user. Also, chat messages do not share many of the design considerations of replicated 3D state (like extreme bandwidth intensity, different message delivery guarantees) so differently-designed protocols make sense.

Having said that, there are interesting ways to integrate chat with the 3D scene, such as chat balloons above avatar heads or chat that is "heard" only within a certain distance from the speaker, that suggest some connection between the processing of the two

protocols. We think that these cases can be supported by encoding SWAMP entity names into fields of XMPP messages.

## 7. SUMMARY

This paper has described a simple protocol, SWAMP, that is sufficient to underlie many applications of multi-user X3D and at the same time has been designed to be practical over the Internet. SWAMP adopts X3D's entity-state model for straightforward integration with X3D browsers and their existing scene access APIs. SWAMP supports both reliable byte-stream (TCP) and fast datagram (UDP) transports. SWAMP is flexible with respect to computational architectures and scene-graph integration schemes.

SWAMP's content message design and encoding supports extremely compact transmission of shared-state information, which is important over internetworks with limited, heterogeneous available bandwidth. SWAMP includes successful design features from many previous systems, emphasizing integration of these features into a flexible, open protocol.

## 8. ACKNOWLEDGMENTS

This work was supported independently by Dr. Weber, and by Media Machines, Inc.

## 9. REFERENCES

- BLAXXUN TECHNOLOGIES 2006, *Developer Information*: <http://developer.blaxxun.com/developer/index.html>
- IEEE 1993. *IEEE standard for information technology – protocols for distributed simulation applications: entity information and interaction*. IEEE Standard 1278-1993. New York: IEEE Computer Society, 1993.
- KIMBALL SOFTWARE 2006, *ABNet2 Software*: <http://kimballsoftware.com/abnet/>
- PARISI, T. 2006. *The Flux X3D Player*, Media Machines Inc., <http://www.mediamachines.com>
- SINGHAL, S., AND ZYDA, M. 1999. *Networked Virtual Environments: Design and Implementation*. New York: ACM SIGGRAPH, ACM Press, Addison-Wesley.
- SINGHAL, S. K., NGUYEN, B. Q., REDPATH, R., FRAENKEL, M., AND NGUYEN, J. 1997. InVerse: Designing an interactive universe architecture for scalability and extensibility. In *Proceedings of the Sixth IEEE International Symposium on High-Performance Distributed Computing (HPDC)*. IEEE Computer Society, Portland OR.
- THORNE, CHRIS 2006. *New Network Sensor Node Proposal*: <http://planet-earth.org/x3d/networkSensorProposal.html>
- THE WEB3D CONSORTIUM. 2004. *The X3D framework & SAI — ISO/IEC FDIS (Final Draft International Standard) 19775:200x*. <http://www.web3d.org/x3d/specifications/>
- WHITE, S.F. 1998. *VNet*. Student Work, University of Waterloo, information at <http://members.optushome.com.au/miriame1/files/FAQ.html>.