ISI-TR-2005-611

The X-Bone API

Yi-Hua Edward Yang, Joe Touch, Gregory G. Finn
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292-6695 U.S.A.

{yeyang, touch, finn } @isi.edu

Dec. 9, 2005

ABSTRACT[1]

This paper describes the Application Programming Interface (X-Bone API) version 2.0 of the current X-Bone Version 3.2 release. It covers both syntax and semantics of the API. X-Bone is a Virtual Internet system that dynamically deploys and manages Internet overlays. The X-Bone API, written in XML (Extensible Markup Language), is the method by which the X-Bone system receives commands and returns responses from/to users to determine these overlays and their properties.

Keywords

Virtual network, Virtual Internet, overlay network, application programming interface (API), extensible markup language (XML).

1. Introduction

An X-Bone Virtual Internet system [5][6] is a collection of Resource Daemons (RD) managed by one or more Overlay Managers (OM). The X-Bone API enables users to communicate with an OM, over an assigned TCP port 265, to configure the RDs (over another assigned TCP port 2165) in the X-Bone VI system on the user's behalf. A "user" here can be a human at a web-based GUI or a text console, or an external program that acts as one, exchanging API messages with the OM. The API messages carry user commands and OM replies for overlay creation, destruction, monitoring, and resource discovery.

An overlay network in the X-Bone VI system consists of a connected graph of virtual nodes interconnected by a number of point-to-point virtual links. A virtual node here is an abstraction; it may denote a *simple* node of a single RD, or a *meta* node of a lower-layer, recursed overlay network. A virtual link connects pairs of virtual nodes with unique endpoint addresses within the overlay network. Each virtual link incorporates two layers of tunneling to emulate both link and network layers for complete virtualization.

The role of the X-Bone API in the management of an X-Bone VI system can be illustrated in Figure 1. The user initiates the command-execute-reply process by issuing commands as API messages to the OM over an SSL-protected stream on TCP port 265 in a particular, desired X-Bone VI system [3]. The OM translates the API message into configuration instructions and dispatches them to reachable RDs via a separate protocol: xb-ctl. The RDs evaluate these instructions, deciding whether to accept and perform them, decline them, or ignore them, and send appropriate responses back to the OM. The OM collects responses and packs them into a reply message also defined by the X-Bone API. The message is then sent back to the user.

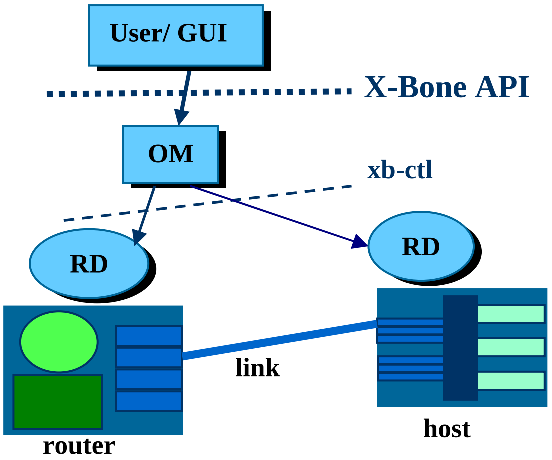

Figure 1 Structure of an X-Bone Virtual Internet.

This report focuses on the design and structure of the X-Bone API. Section 2 lists keywords and conventions used throughout this document; Section 3 describes the syntax and Section 4 explains the semantics of all the elements in the API except the X-Bone Overlay

[1] This material is based upon work supported by the National Science Foundation under Grant No. <STI-XTEND: ANI-0230789>. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Language (XOL), whose syntax and semantics are explained separately in Section 5. Section 6 descibes network recursion support, with security issues in Section 7 and future work in Section 8. This document supersedes and obsoletes ISI-TR-2001-549 [2].

2. Reserved Tokens and Conventions

This section describes the fundamental properties of the X-Bone API XML-based syntax. Before proceeding to the complete language description in Section 3, this section lists the reserved tokens and conventions of other user tokens in the language.

2.1 Reserved Tokens

The element and attribute names are the reserved tokens used by the API. Use of these tokens, except as indicated by the language, must be avoided.

`argstring`	`node_def`
`create_overlay*`	`overlay`
`credential`	`overlay_status`
`destroyall_overlays*`	**`property`**
`destroy_overlay*`	`renames`
`discover_daemons*`	`synonym`
`endpoint`	**`tag`**
`define_prop`	`tunnel`
`ident`	**`value`**
`iface`	`version`
`link`	`vnode`
`list_overlays*`	`xol_program`
`node`	

Table 1 Reserved Tokens

In Table 1, tokens with a star (*) have a corresponding token with "`_reply`" appended, e.g., "`create_overlay_reply`". These entries are omitted for clarity and brevity.

Further, some of reserved tokens are part of the conventional XML DTDs, and are not specific to the X-Bone, but are listed there as well; shown in bold.

2.2 String Conventions

User-supplied strings in the API follow the CDATA syntax restriction of XML. Strings are delimited by double quotes, "…". Some special characters (&, <, >, ', ") inside a CDATA string must be quoted in their escaped form, as shown in Table 2.

Where the semantic nature of a string is more restrictive, as in E-mail or DNS names, those particular restrictions are applied as well. All strings and tokens are verified for such restrictions.

Character	Escaped Quotation
`&`	`&amp;`
`<`	`&lt;`
`>`	`&gt;`
`'`	`&apos;`
`"`	`&quot;`

Table 2 Escape characters

2.3 Presentation Conventions

2.3.1 Property Elements

For convenience and simplicity, in this document the `property` elements are referred to by their `tag` attributes. For example, the `property` element with `tag` attribute "user_name" is referred to as the "user_name property."

2.3.2 Message Naming

Every message in the API is either a command message (sent from user to OM) or a reply message (sent from OM to user). The core of a message is one of the 13 *command-reply elements* (Sec. 3.3). Every API message must contain exactly one command-reply element. The purpose of a message is specified by the command-reply element it contains.

For this reason and for simplicity, in this document an API message is named by the type of command-reply element it possesses. For example, an API message containing the `list_overlays_reply` element is called a "list_overlays_reply message." Also, an API message containing a command-carrying element is called a "command message," while a message containing a reply-carrying element is called a "reply message."

2.3.3 Element Naming

Many elements have an `ident` attribute that uniquely identifies them without their containing parents. Specifically, the following XOL elements have this kind of `ident` attribute:

- Node_def
- Iface
- Vnode
- Link

For simplicity, these elements are named by their respective `ident` attribute values. For example, the `vnode` element with “router_0” as its `ident` attribute value is called the “router_0” `vnode` element.

3. X-Bone API Syntax

The X-Bone API syntax consists of grammar rules governing message exchanges across a well-known, privileged TCP port (265) used by X-Bone OMs, where the stream is protected by SSL [3]. All messages in the API adapt a unified XML structure, starting with the sequence of XML Declaration, a Document Type Declaration (DTD), and one `xbone` element, in that order. The current X-Bone (release 3.2) implements X-Bone API version 2.0 (api-2.0.dtd) [1].

The XML elements in X-Bone API can be divided conceptually into four element categories: *common-purpose elements*, *command-reply elements*, *overlay-description elements*, and *XOL elements*. This section steps through the syntax of the first 3 element categories. Syntax of the XOL elements is described in Section 5.1.

3.1 Document Type Definition

Below is the DTD of X-Bone API version 2.0 [1].

```
<?xml version="2.0" ?>

<!ELEMENT xbone (credential?, command)>
<!ATTLIST xbone
    version CDATA #REQUIRED
    release CDATA #REQUIRED>

<!ELEMENT credential (property+)>

<!ELEMENT command
         (create_overlay_reply |
          create_overlay |
          list_overlays_reply |
          list_overlays |
          overlay_status_reply |
          overlay_status |
          discover_daemons_reply |
          discover_daemons |
          destroy_overlay_reply |
          destroy_overlay |
          destroyall_overlays_reply |
          destroyall_overlays |
          error_reply)>

<!ELEMENT create_overlay
         (property+, xol_program)>
<!ELEMENT create_overlay_reply
         (property+, node*)>

<!ELEMENT list_overlays EMPTY>
<!ELEMENT list_overlays_reply
         (property?, argstring*)>

<!ELEMENT overlay_status (property+)>
<!ELEMENT overlay_status_reply
         (property+, node*)>

<!ELEMENT discover_daemons_reply
         (property+, node*)>
<!ELEMENT discover_daemons (property+)>

<!ELEMENT destroy_overlay (property+)>
<!ELEMENT destroy_overlay_reply
         (property+)>

<!ELEMENT destroyall_overlays EMPTY>
<!ELEMENT destroyall_overlays_reply
         (property+)>

<!ELEMENT error_reply (property+)>

<!ELEMENT argstring EMPTY>
<!ATTLIST argstring
    value CDATA #REQUIRED>

<!ELEMENT node (property+, tunnel*)>

<!ELEMENT tunnel (property+)>

<!ELEMENT xol_program
         (define_prop*,node_def+,vnode)>
<!ATTLIST xol_program
    version CDATA #REQUIRED>

<!ELEMENT define_prop (property)>
<!ATTLIST define_prop
    synonym CDATA #REQUIRED>

<!ELEMENT node_def
         (iface+, vnode*, link*,
          property*, application*)>
<!ATTLIST node_def
    ident CDATA #REQUIRED>

<!ELEMENT iface (renames|property*)>
<!ATTLIST iface
    ident CDATA #REQUIRED>

<!ELEMENT renames (endpoint, property*)>

<!ELEMENT vnode (property*)>
<!ATTLIST vnode
    ident CDATA #REQUIRED
    type  CDATA #REQUIRED>

<!ELEMENT link
         (endpoint, endpoint,
          property*)>
<!ATTLIST link
    ident CDATA #REQUIRED>

<!ELEMENT property EMPTY>
<!ATTLIST property
    tag CDATA #REQUIRED
    value CDATA #IMPLIED>

<!ELEMENT endpoint EMPTY>
<!ATTLIST endpoint
    node CDATA #REQUIRED
    iface CDATA #REQUIRED>

<!ELEMENT application EMPTY>
<!ATTLIST application
    program  CDATA #REQUIRED
    script   CDATA #REQUIRED
    checksum CDATA #IMPLIED
    suid     CDATA #IMPLIED
    nodes    CDATA #IMPLIED
    ifaces   CDATA #IMPLIED>
```

3.2 Common-Purpose Elements

The *common-purpose elements* (Table 3) are used to form the basic structure of any message in the API.

Element name	Attributes	Sub-elements
xbone	version, release	credential?, command
credential	(none)	property+
command	(none)	(One of the elements in Table 4.)
property	tag, value	EMPTY

Table 3 Common-purpose elements

The root of the API is the `xbone` element, which is described by the `version` and `release` attributes. The `xbone` element contains an optional `credential` element and a single `command` element. The `credential` element contains at least one property. The `command` element contains one of the 13 *command-reply elements* (see the sub-section below).

The `property` element is used pervasively in almost all other elements in the API to associate a tag with a value, with both tag and value as its attributes. The `tag` and `value` attribute pair can be either a command parameter or a return value; it usually describes a semantic aspect of the property's parent element.

3.3 Command-Reply Elements

There are 13 different *command-reply elements*, each of which may be the sole child of the enclosing `command` element (see the sub-section above). There are six command-carrying elements and six corresponding reply-carrying elements plus one error_reply element, shown in Table 4.

Except for the `destroyall_overlays` and `list_overlays` elements, which need no parameter to work, all other 11 command-reply elements contain at least one property. In command-carrying elements, a property represents a parameter sent along with the user command; in reply-carrying elements, a property represents a return value replied from the OM.

The `create_overlay` element contains as its last child the `xol_program` element, which is the root of XOL (Sec. 5.1) and describes the complete overlay structure to create.

The `create_overlay_reply`, `discover_daemons_reply`, and `overlay_status_reply` elements may also contain one or more `node` elements describing the virtual nodes replied from the OM. The `list_overlays_reply` element uses an `argstring` element to return the list of overlays from OM to user. Both `node` and `argstring` elements are explained in the next sub-section (Section 3.4).

Element name	Sub-elements
create_overlay	property+, xol_program
create_overlay_reply	property+, node*
destroy_overlay	property+
destroy_overlay_reply	property+
destroyall_overlays	EMPTY
destroyall_overlays_reply	property+
discover_daemons	property+
discover_daemons_reply	property+, node*
list_overlays	EMPTY
list_overlays_reply	property?, argstring*
overlay_status	property+
overlay_status_reply	property+, node*
error_reply	property+

Table 4 Command-reply elements

3.4 Overlay-Description Elements

The three *overlay-description elements* (`node`, `tunnel`, and `argstring`) are used by a number of reply messages to describe the overlay and its nodes and tunnels, and are described in Table 5.

Element name	Attributes	Sub-elements
argstring	Value	EMPTY
node	(none)	property+, tunnel*
tunnel	(none)	property+

Table 5 Overlay-description elements

The `argstring` element enlists a `value` attribute to return a string of overlay names, each separated by a comma.

The `node` elements describe the virtual nodes of a returned overlay network. A virtual node can have any number of tunnels, each represented by a `tunnel` element. Detail configuration of the virtual nodes and tunnels are carried by the properties enclosed in the `node` and `tunnel` elements, respectively.

4. X- Bone API Semantics

This section explains the semantics of the first three element categories of the API: *common-purpose* *elements*, *command-reply elements*, and *overlay-description elements*. Semantics of the *XOL elements* are explained in Section 5.2, after the XOL syntax.

The semantics of the API reside in the name, attributes, and sub-elements (including properties) of the elements. Consequently, this section and Section 5.2 explains each element by describing what it is (the name), what attributes (if any) it has, and the sub-elements it may contain, in that order. The properties of an element are listed at the end of each sub-section.

4.1 Common-Purpose Elements

Fundamentally, an X-Bone API message is either a user command sent to the OM or an OM reply sent back to the user. The *common-purpose elements* describe the type and structure of the message and the identity of the issuing/receiving user.

4.1.1 Property

The `property` element is one of the most useful elements in the API; it is used by almost all other elements to carry a parameter or a return value. By enclosing multiple `property` elements, multiple parameters or return values can be passed for the parent of those property elements.

```
<!ELEMENT property EMPTY>
<!ATTLIST property tag   CDATA #REQUIRED
                   value CDATA #IMPLIED>
```

The `tag` and `value` attributes, which represent respectively the name and the content of a parameter or a return value, usually depend greatly upon the particular context where the property appears. While some property tag/value pairs are required, in general, the X-Bone system has default behaviors for missing optional property tags/values. Legal property tags and values are listed at the end of a sub-section for the semantics of each property-containing element.

4.1.2 Xbone

The `xbone` element is the root of the API and encloses all other elements in the message.

```
<!ELEMENT xbone ( credential?, command )>
<!ATTLIST xbone version CDATA #REQUIRED
                release CDATA #REQUIRED>
```

The `version` attribute contains the version number of the API's DTD being used. The `release` attribute contains the current X-Bone system release number to which this message applies to. Both version and release values take the form of "number.number". A valid example is:

```
<xbone version=”2.0” release=”3.2”>
```

In a command message, the `xbone` element must contain a `credential` sub-element, which identifies the sender of the command, and a `command` sub-element, which specifies the command itself. For a reply message, however, the `credential` sub-element is optional and generally not produced, while the `command` sub-element contains the reply contents.

4.1.3 Credential

The `credential` element identifies the command issuer to the X-Bone system. The credentials, including user name, email, and authentication type, should have been extracted from a trustworthy source, such as the issuer's PKI certificate (as is performed by the X-Bone web GUI), and put securely into the properties of the `credential` element.

```
<!ELEMENT credential ( property+ )>
```

The following property tag/value pairs are all required:

- The user_name property specifies the name of the issuer (user invoking the command).
- The user_email property specifies the email address of the issuer.
- The auth_type property specifies the type of authentication used to validate the user. Currently, only 'x509' is supported.

4.1.4 Command

The `command` element is a simple element that wraps around one of the 13 *command-reply elements* (Sec. 4.2), including 6 command-carrying elements, 6 reply-carrying elements, and an `error_reply` element.

```
<!ELEMENT command (
  create_overlay |
  create_overlay_reply |
  list_overlays |
  list_overlays_reply |
  overlay_status |
  overlay_status_reply |
  discover_daemons |
  discover_daemons_reply |
  destroy_overlay |
  destroy_overlay_reply |
  destroyall_overlays |
  destroyall_overlays_reply |
  error_reply  )>
```

Beside these sub-elements, the `command` element has neither attribute nor property associated to it.

4.2 Command-Reply Elements

The *command-reply elements* include both command-carrying elements that carry user commands to OM and reply-carrying elements that carry OM's replies to user. All command-reply elements appear inside the command element, which in turn appears inside the xbone element (root of the API).

From the semantic point of view, the command-reply elements can be divided into five categories: overlay creation (Secs. 4.2.1 & 4.2.2), overlay destruction (Sec. 4.2.3 – 4.2.6), resource discovery (Sec. 4.2.7 & 4.2.8), overlay status query (Sec. 4.2.9 – 4.2.12), and error reply (Sec. 4.2.13).

4.2.1 Create_overlay

The create_overlay element is used inside a command message to specify the user's intent to create an overlay network.

```
<!ELEMENT create_overlay
        ( property+, xol_program )>
```

The xol_program element is the root of the X-Bone Overlay Language (sec. 5); it describes the complete overlay network structure the OM is asked to create.

The following properties define certain environmental information to be applied to the xol_program element:

- The "address_server" property value must be a host address. This property is optional; the default is to use the address server configured into the X-Bone.
- The "address_server_port" property value must be a port number. This property is optional; the default is to use the port number configured into the X-Bone.
- The "creator_email" property value should contain a properly formatted e-mail address. This property is required.
- The "creator_name" property value is an unrestricted CDATA string. This property is required.
- The "manager" property value must be a host name. This property is optional. The default is to use the manager configured into the X-Bone.
- The "manager_port" property value must be a port number. This property is optional. The default uses the port configured into the X-Bone.
- The "overlay_name" property has value as a CDATA string. This string should follow DNS naming conventions. This property is required.
- The "topology" property value may be one of the following: "ring", "linear", "star" or "custom". This value specifies the desired topology in which to create the overlay. The detail of a "custom" topology is further specified by the structure and contents of the xol_program element. This property pair is required.
- The "custom_hostlist" property is a list of white space-separated host addresses of RDs on which to create the overlay network.
- The "ldap", "attrvals", and "scope" properties are used to configure an LDAP query message to find the RDs to participate in the overlay creation from an LDAP server.

4.2.2 Create_overlay_reply

The OM replies a create_overlay_reply message to the user after overseeing RDs on overlay creation.

```
<!ELEMENT create_overlay_reply
        ( property+, node* )>
```

The node elements, explained in Sec. 4.3.1, represent and describe the nodes chosen to participate in the overlay.

- The "overlay_name" property value is the name of the overlay as passed to the OM by the original create_overlay command.
- The "dns" property value contains the base DNS name of the node. This property element is not present if DNS naming was not requested for the overlay created.
- The "routing" property indicates whether a static routing table is used or a dynamic routing daemon is enabled for the overlay network.
- The "IPsec_encryption" property value shows the encryption method used by the nodes in the overlay.
- The "IPsec_authentication" property value show the authentication method used by the nodes in the overlay.

4.2.3 Destroy_overlay

A destroy_overlay message indicates the user's intent to destroy a previously created overlay.

```
<!ELEMENT destroy_overlay (property+)>
```

- The "overlay_name" property specifies the name of the overlay to destroy. The name should match one that passed to the OM by a previous create_overlay command. This property is required.

4.2.4 Destroy_overlay_reply

The OM replies a destroy_overlay_reply message to the user if RDs successfully destroy the user-specified overlay. Otherwise, an error_reply message (4.2.13) is replied.

```
<!ELEMENT destroy_overlay_reply
        ( property+ )>
```

- The “overlay_name” property specifies the name of the overlay as passed to the OM by the original destroy_overlay command.

4.2.5 Destroyall_overlays

A destroyall_overlays message tells the OM to destroy all overlays it manages.

```
<!ELEMENT destroyall_overlays EMPTY>
```

4.2.6 Destroyall_overlays_reply

The OM replies a destroyall_overlays_reply message to the user if RDs successfully destroy all the overlays it manages. Otherwise, an error_reply message is replied to the user.

```
<!ELEMENT destroyall_overlays_reply
        ( property+ )>
```

- The “message” property carries the message the OM returns to the user after the destruction of the overlays.

4.2.7 Discover_daemons

A discover_daemons message asks the OM to return all available RDs managed by it.

```
<!ELEMENT discover_daemons (property+)>
```

- The “creator_email” property should contain a properly formatted e-mail address. This property is required.
- The “creator_name” property is an unrestricted CDATA string. This property is required.
- The “search_radius” property must be a positive integer. This property is optional. The default is to use the hop-count configured into the X-Bone.
- The “timeout” property must be a positive integer. This is a required property.
- The “custom_hostlist” property value is a list of white space-separated IP addresses specifying the hosts from which to discover RDs.
- The “ldap”, “attrvals”, and “scope” properties configure an LDAP query message to discovery daemons from an LDAP server.

4.2.8 Discover_daemons_reply

The OM replies a discover_daemons_reply message if some resource daemons are successfully found by a previous discover_daemons command. Otherwise, an error_reply message is replied.

```
<!ELEMENT discover_daemons_reply
        ( property+, node* )>
```

The `node` sub-elements describe the states of the resource daemon returned from a previous discover_daemons command.

- The “creator_email” property should contain a properly formatted e-mail address.
- The “creator_name” property is an unrestricted CDATA string.

4.2.9 List_overlays

A list_overlays message asks the OM to list all the overlays managed by it.

```
<!ELEMENT list_overlays EMPTY>
```

4.2.10 List_overlays_reply

The OM replies a list_overlays_reply message if a list of overlay names was generated from a previous list_overlays command. Otherwise, an error_reply message is replied.

```
<!ELEMENT list_overlays_reply
        ( property?, argstring* )>
```

The `argstring` sub-element contains the list of overlay names returned to the user. The `property` sub-element, although listed in the DTD, is not used by the current X-Bone release (Version 3.2).

4.2.11 Overlay_status

An overlay_status message queries the OM for the status of an overlay.

```
<!ELEMENT overlay_status (property+)>
```

- The “overlay_name” property is the name of the overlay as passed to the RD by a previous create_overlay command. This property required.
- The “search_radius” key value must be a positive integer. This property is optional. The default is to use the hop-count configured into the X-Bone.
- The “timeout” key value must be a positive integer. This property is optional. The default behavior is to use the timeout configured into the X-Bone.

4.2.12 Overlay_status_reply

The OM replies an overlay_status_reply message to the user if it successfully collects status information about the specified overlay. Otherwise, an error_reply message is replied.

```
<!ELEMENT overlay_status_reply
        ( property+, node* )>
```

The child `node` elements describe the states of the virtual nodes in the overlay specified in a previous overlay_status command message.

- The “creator_email” property contains a properly formatted e-mail address of the user who created the named overlay.
- The “creator_name” property contains the name of the user who created the named overlay.
- The “IPsec_encryption” key value contains the encryption method used by the named overlay.
- The “IPsec_authentication” key value contains the authentication method used by the named overlay.
- The “dns” property value contains the base DNS name of the node. This property element is not present if DNS naming was not requested for the overlay created.
- The “routing” property indicates whether a static routing table is used or a dynamic routing daemon is enabled for the overlay network.
- The “overlay_name” property is the name of the overlay network whose status is being replied. This value matches the “overlay_name” property originally used to create the overlay network.

4.2.13 Error_reply

This element is used by the X-Bone system to indicate an error that has occurred. An error_reply message is generated by the OM in reply to any command message in case of errors.

```
<!ELEMENT error_reply (property+)>
```

- The “command” property contains the command-carrying element name which caused the error.
- The “error” property contains the error text.

4.3 Overlay-Description Elements

The three *overlay-description elements* (`node`, `tunnel`, `argstring`) are used only in reply messages to describe overlays. They are always sent from the OM to the user, but never from the user to the OM. Specifically, the `node` and `tunnel` elements are used by reply messages of overlay creation, daemon discovery, and (overlay) status query; the `argstring` element is used only by the list_overlays_reply message.

4.3.1 Node

The `node` element is used in two contexts. When used inside a create_overlay_reply or overlay_status_reply message, the `node` element describes the state of a virtual node in an overlay network. When used inside a discover_daemons_reply message, the `node` element describes an RD in the X-Bone system.

```
<!ELEMENT node ( property+, tunnel* )>
```

The following properties are common to all types of `node` elements:

- The “class” property has value either “simple” or “meta”. A node of “simple” class is either a single host or a single router; a node of “meta” class is an overlay network built from other virtual nodes.
- The “hostname” property value specifies the hostname of the returned virtual node or RD.
- The “os” and “os_version” properties describe the OS running the virtual node or the RD.

As a child element of `create_overlay_reply` (sec. 4.2.2) or `overlay_status_reply` (sec. 4.2.12), the `node` element can have the following properties:

- The “ip” property value is the string representation of the IPv4 or IPv6 numeric address of the node. An IPv4 address is represented as four-field dotted decimal: “128.9.160.30”. An IPv6 address is represented as colon-separated hexadecimal fields: “2001:470:1f00:1019:207:e9ff:fe09:44ac”.
- The “status” property value can be either “up” or “down”, and describes the status of the node for an overlay. A node with a “down” status either could not be reached or did not respond in time.
- The “type” property has value either “host” or “router”, depending whether the virtual node is a host or a router.
- The “vname” property value specifies the `ident` attribute of the `vnode` element (sec. 5.2.4) that created the virtual node when constructing the overlay network.

As a child element of `discover_daemons_reply`, the `node` element can have the following properties:

- The “app_addr” and “app_addr6” property values are IPv4 and IPv6 addresses, respectively, visible to applications outside the overlay.

- The “ctl_addr” and “ctl_addr6” property values are IPv4 and IPv6 addresses, respectively, that are used by OM (meta node) for sending and receiving control messages.
- The “dns” property value contains the base DNS name of the node. This property element is not present if DNS naming was not requested for the overlay in which the node resides.
- The “ipproto” property specifies the IP protocol the discovered daemon operates on.
- The “IPsec” property value can be either “yes” or “no”, depending on whether IPsec is used to communicate with the node.
- The “kernel” property describes the kernel version of the operating system.
- The “node_type” property has value either “meta”, “router”, or “host”. A “meta” node represents an OM. A “host” node is an RD with one active interface, while a “router” node is an RD with more than one active interface.
- The “overlays” property value indicates the number of overlays in which this node is currently participating.
- The “routing” property has value either “yes (dynamic)” or “no (static)”.
- The “tunnel” property value counts the number of active tunnels (interfaces) of the node.
- The “xol_ver” property value is the XOL version the node currently uses. The format is “integer.alphanumeric”. This property is only meaningful for meta nodes.

4.3.2 Tunnel

A `tunnel` element describes the status of a tunnel of the containing `node` element. It can only appear inside a `node` element.

```
<!ELEMENT tunnel ( property+ )>
```

- The “local_ip_address” property value is a numeric form IPv4 or IPv6 address of the local end of this tunnel of this node.
- The “remote_ip_address” property value is a numeric form IPv4 or IPv6 address of the remote end of this tunnel of this node.
- The “status” property describes the status of the tunnel for the containing node. Its value can be either “up” or “down”.

4.3.3 Argstring

The `argstring` element only appears as a sub-element of the `list_overlays_reply` element. It is an empty element described by the `value` attribute.

```
<!ELEMENT argstring EMPTY>
<!ATTLIST argstring
          value CDATA #REQUIRED>
```

The `value` attribute is a string of overlay names separated by a comma and optional white spaces. This string represents the list of overlays replied for a previous list_overlays command. Valid examples of the attribute are “test_ring” and “neta, netb, netc”.

5. X- Bone Overlay Language

The X-Bone Overlay Language (XOL) is a self-contained overlay-description language inside the X-Bone API. It can describe an entire multi-level, recursive overlay structure. The *XOL elements* are used exclusively in a `create_overlay` element to describe the structure and properties of the overlay network.

5.1 XOL Elements Syntax

The *XOL elements* (Table 6) make up the X-Bone Overlay Language which defines an overlay network.

Element name	**Attributes**	**Sub-elements**
`xol_program`	version	define_prop*, node_def+, vnode
`define_prop`	synonym	property
`node_def`	ident	iface+, vnode*, link*, property*, application*
`iface`	ident	renames | property*
`vnode`	ident, type	property*
`link`	ident	endpoint, endpoint, property*
`endpoint`	iface, node	EMPTY
`renames`	(none)	endpoint, property*
`application`	program, script, checksum, suid, nodes, ifaces	EMPTY

Table 6 XOL elements

The XOL is a self-containing language: an XOL element only contains other XOL elements (or the property element, which is used throughout the API);

also, other than root xol_program, XOL elements are never used outside XOL.

5.2 XOL Elements Semantics

5.2.1 Xol_program

The xol_program element is the root of XOL. It contains elements for property synonyms and node definitions, plus a closing vnode element (sec. 5.2.6) representing the entire overlay.

```
<!ELEMENT xol_program
        ( define_prop*, node_def+,
          vnode )>
```

5.2.2 Define_prop

A define_prop element introduces a synonym for a single property element. This allows the XOL programmer to define a commonly used property key/value pair, assign a synonym to it, and then use that synonym word in subsequent property elements to imply the key/value pair.

```
<!ELEMENT define_prop (property)>
```

```
<!ATTLIST define_prop
          synonym CDATA #REQUIRED>
```

Below is an example of the define_prop element:

```
  <define_prop synonym="1Gps">

    <property tag="speed"
            value="1000000000"/>

  </define_prop>
```

The synonym can then be used in a subsequent property element like this:

```
  <property tag="1Gps"/>
```

5.2.3 Node_def

A node_def element defines the *type* for a virtual node. It does not create any instance of virtual node per se, but provides a template for virtual node instantiation by a later vnode element.

```
<!ELEMENT node_def
        ( iface+, vnode*, link*,
          property*, application* ) >
<!ATTLIST node_def ident CDATA #REQUIRED>
```

The ident attribute of a node_def element uniquely identifies it within the enclosing xol_program. This attribute value is used by the later vnode element (sec. 5.2.4) which instantiates a virtual node of type defined by this node_def.

When the node_def element defines a type for a *simple* node, it cannot contain any vnode or link sub-elements. A node_def element defines a type for a *meta* node by including multiple vnode and link as child elements.

The node_def element for a virtual host often has only one iface sub-element; the node_def for a virtual router must have multiple iface sub-elements.

- The “address_type” property value may be one of “IPv4” or “IPv6”. All virtual nodes in an overlay network should use the same address type. This property is required.
- The “dns” property value may be either “yes” or “no”. When set to “yes”, this property enables the “name_server” and “name_server_port” properties. All virtual nodes in an overlay network should have the same dns value. This property is required.
- The “dynamic_routing” property value must be either “yes” or “no”. This property is required.
- The “IPsec_encryption” property value must be an encryption method supported by the X-Bone. Currently, only “des”, “3des”, and “none” are supported. This property is optional. The default is “none”.
- The “IPsec_authentication” property value must be an authentication method supported by the X-Bone. Currently, only “sha1”, “md5”, and “none” are supported. This property is optional. The default is “none”.
- The “name_server” property value must be a host name. This property is optional, and may appear only if the “dns” property has value “yes”. The default is to use the name server configured into the X-Bone.
- The “name_server_port” property value must be a port number. This property is optional, and may appear only if the “dns” property has value “yes”. The default is to use the name server port number configured into the X-Bone.
- The “os” property specifies the desired operating system running the virtual node. It affects only simple nodes, and is ignored by meta nodes. Currently, only “FreeBSD” and “Linux” are recognized. This property is optional. The default is “FreeBSD”.

5.2.4 Iface

The iface element defines an interface for the enclosing node_def element. An interface is an exported contact point of a virtual node. The iface element describes an interface in the same way as the node_def element describes a virtual node.

```
<!ELEMENT iface ( property* | renames )>
<!ATTLIST iface ident CDATA #REQUIRED>
```

The `ident` attribute of an `iface` element gives the interface a name by which it is uniquely identified within the enclosing `node_def` element. Optional information associated with an `iface` may reside in its `property` elements, although currently (X-Bone release 3.2) no property is defined for the `iface` element.

When an `iface` contains a single `renames` element (sec. 5.2.9), it acts as an exported alias to the interface described by the `renames` element. In this case, the `iface` element must be enclosed inside a `node_def` element that defines the type for a *meta* node.

5.2.5 Vnode

A `vnode` element declares an instance of a virtual node within an overlay network.

```
<!ELEMENT vnode (property*)>
<!ATTLIST vnode ident CDATA #REQUIRED
                type  CDATA #REQUIRED>
```

The `ident` attribute of a `vnode` element gives it a name by which it can be uniquely identified within the enclosing `node_def` element.

The `type` attribute of a `vnode` must match the `ident` attribute of a previously defined `node_def`, which is used as a template for this virtual node instantiation.

Any property of the `node_def` element can be used inside the `vnode` element. The properties of a `vnode` element add to and override any property implied by the template `node_def` element.

5.2.6 Closing vnode element

The last `vnode` of an `xol_program` element instantiate a virtual node that represents the entire overlay network.

The `ident` attribute of the closing `vnode` is used for naming the overlay network. If DNS is enabled, the `ident` attribute also becomes part of the DNS name associated with the overlay network.

The `type` attribute specifies the `node_def` element used as a template for the entire overlay network. This `node_def` element must define a *meta* node and contain more than one `vnode` and `link` sub-elements.

5.2.7 Link

A `link` element represents a tunnel that connects two virtual nodes.

```
<!ELEMENT link ( endpoint, endpoint,
                property* )>
<!ATTLIST link ident CDATA #REQUIRED>
```

The `ident` attribute of a `link` element gives it a name by which it can be uniquely identified within the enclosing `node_def` element.

The two `endpoint` elements (sec. 5.2.8) inside the `link` element specify the endpoints of the tunnel represented by this `link` element.

5.2.8 Endpoint

An `endpoint` element defines one end of a tunnel. Each `endpoint` element corresponds to an interface on a virtual node, and can be associated to at most one tunnel (i.e., be used in at most one `link` element).

```
<!ELEMENT endpoint EMPTY>
<!ATTLIST endpoint node CDATA #REQUIRED
                   iface CDATA #REQUIRED>
```

An `endpoint` is uniquely identified by its `node` and `iface` attributes. Its `node` attribute must match the `ident` of some previously defined `vnode`; its `iface` attribute must refer to the `ident` of an `iface` within the `node_def` that defines the type of that `vnode`.

5.2.9 Renames

The `renames` element indicates an internal network endpoint to be an ‘exported’ interface.

```
<!ELEMENT renames ( endpoint, property*
)>
```

When a `node_def` element defines a type of virtual network, the `iface` elements defined within the `node_def` correspond to the exported interfaces by which this network may be connected to other networks. These interfaces are mapped onto interfaces on the network’s constituent virtual nodes. Each `renames` element establishes one such mapping.

For example,

```
<iface ident="exp_0">
  <renames>
    <endpoint node="router_0”
              iface="if_3"/>
  </renames>
</iface>
```

In this example, the `iface` element identifies that “exp_0” is an exported interface mapped to the interface “if_3” of virtual node “router_0”. Note that both the “exp_0” `iface` and “router_0” `vnode` elements belong to the same `node_def` element; while the “if_3” `iface` element belongs to the `node_def` that *defines* the “router_0” `vnode` element (the `node_def` specified by the `type` attribute of the “router_0” `vnode` contains the “if_3” `iface`).

Any property specified in the `renames` element is applied to the parent `iface` element *in addition to* those already defined for the underlying endpoint element.

5.2.10 Application

An application element specifies what to run once a virtual node defined by the containing node_def is instantiated. More than one application may be associated with a given node_def.

```
<!ELEMENT application EMPTY>
<!ATTLIST application
          program  CDATA #REQUIRED
          script   CDATA #REQUIRED
          checksum CDATA #IMPLIED
          suid     CDATA #IMPLIED
          nodes    CDATA #IMPLIED
          ifaces   CDATA #IMPLIED>
```

The program attribute names the program to be run as the application. The script attribute contains the script to be passed to the program when it is started. The checksum attribute is optional and is a hexadecimal string of the checksum of the script. The suid attribute specifies the uid the application script will use.

The nodes attribute specifies the types of nodes, hosts or routers that the application will execute upon. The ifaces attribute specifies whether the application script will use all interfaces on a given router or just one.

6. Advanced Features

The X-Bone API version 2.0 includes support on overlay recursion[2]. In an X-Bone Virtual Internet system, overlays recurse by emulating a virtual network as a virtual router in the base network. There are two types of recursions in the X-Bone Virtual Internet: *control recursion* and *network recursion*.

- *Control recursion* allows a compact symbolic representation to be expanded during deployment in order to divide-and-conquer a large, flat network management.
- *Network recursion* is true stacking of a VI on top another VI, where the packets on the upper layer have additional header encapsulation. The hops and nodes inside the recursed (lower) network are not visible in the recursive upper layer. To the upper layer, the recursed network looks exactly like a single-node router.

The X-Bone API allows network overlay recursion through the use of XOL. In XOL, each vnode element is type-defined by a node_def element, which itself may contain other vnode (virtual node) and tunnel (virtual link) elements as its components. In such cases, the node_def element effectively describes the vnode element it defines as a virtual *network*. Network overlay recursion occurs when this virtual-networked vnode element is used as a component inside a node_def element that defines another (higher-layer) virtual-networked vnode element.

Figure 2 illustrate the language structure of XOL that enables network recursion. This kind of recursive language structure is analogous to the struct-type recursion in C programming language, where a struct definition can have declarations of other struct instances inside it.

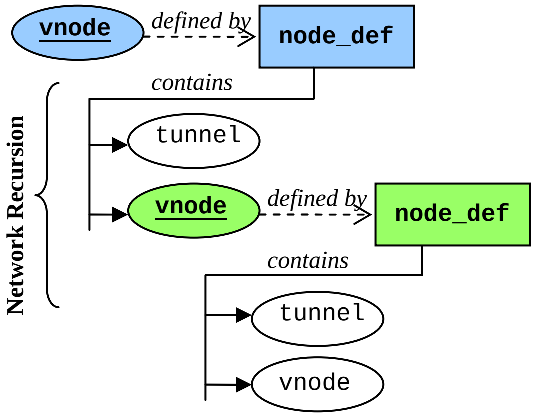

Figure 2 Network overlay recursion in XOL.

7. Security Issues

The contents of the credential element in each message is neither encrypted nor authenticated within the X-Bone API. Since the correctness of the user credential depends on the *ends* of the communication (user and OM), it is neither sufficient nor necessary to offer protection inside the API. To authenticate the user or to encrypt the API message, an out-of-band security mechanism is needed.

The current X-Bone release uses SSL connection between the user and the OM and verifies host identity using the SSL certificate. Any message sent from the host is assumed trustworthy. While this security-binding mechanism seems rather weak, for example, a user on a verified host can easily send an API message with fraudulent credential information to the OM, discussions on improvements to such mechanism is out of the scope of this paper.

[2] The current X-Bone release 3.2 has not fully implemented these recursion capabilities.

8. Future Work

Work is in progress to add peer-to-peer support to X-Bone (P2P-XBone) in order to deploy virtual IP networks with P2P's characteristics such as self-organization and late-binding [7]. Compared to existing application-layer P2P networks, P2P-XBone constructs peer-to-peer network at the network layer and allows end-to-end applications work with existing network and transport protocols. This also gives applications running on P2P-XBone higher forwarding performance and simpler implementation.

P2P-XBone adds a few extensions to the X-Bone API in order to enable P2P-style topology management:

- The `join_overlay` and `leave_overlay` command elements are added to instruct the OM to add/remove a single RD to/from an existing overlay network. The `join_overlay_reply` and `leave_overlay_reply` reply elements are then sent from the OM to the user, who sits at the RD that wishes to join or leave the overlay.
- Two `node_def` element properties, "p2p_port" and "p2p_bootstrap", are added specifically for the operation of the peer-to-peer protocol.

9. References

[1] X-Bone API version 2.0 DTD: www.isi.edu/xbone/software/xbone/api-2.0.dtd

[2] Finn, G.G., Touch, J.D, "X-Bone Application Programmers Interface : X-Bone Overlay Language (XOL) Specification," ISI-TR-2001-549, Nov. 2001.

[3] Hickman, K., "The SSL Protocol," Netscape Communications Corp., Feb. 1995.

[4] Touch, J., Y. Wang, L. Eggert, G. Finn, "A Virtual Internet Architecture," Technical Report ISI-TR-2003-570, USC/ISI, CA, 2003.

[5] Touch, J., S. Hotz, "The X-Bone," Third Global Internet Mini-Conference, Globecom '98. Sydney, Australia, November 1998, pp. 59-68

[6] Touch, J., "Dynamic Internet Overlay Deployment and Management Using the X-Bone," Computer Networks, July 2001, pp. 117-135.

[7] Fujita, N., Touch, J., Pingali V., Wang, Y., "P2P-XBone: A Virtual Network Support for Peer-to-Peer Systems," Technical Report ISI-TR-2005-607, USC/ISI, September 2005.