dispatch	J. Rosenberg	
Internet-Draft	jdrosen.net	
Intended status: Standards Track	C. Jennings	
Expires: April 28, 2011	Cisco	
	M. Petit-Huguenin	
	Stonyfish	
	October 25, 2010	

TOC

Verification Involving PSTN Reachability: The ViPR Access Protocol (VAP)
draft-rosenberg-dispatch-vipr-vap-03

Abstract

Verification Involving PSTN Reachability (ViPR) is a technique for inter-domain SIP federation. ViPR hybridizes the PSTN, P2P networks, and SIP, and in doing so, addresses the phone number routing and VoIP spam problems that have been a barrier to federation. The ViPR architecture uses a server, the ViPR server, which performs P2P and validation services on behalf of call agents, which acts as clients to the server. Such an architecture requires a client/server protocol between call agents and the ViPR server. That protocol, defined here, is called the ViPR Access Protocol (VAP).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as “work in progress.”
This Internet-Draft will expire on April 28, 2011.

Table of Contents

1. Introduction to ViPR
2. Overview of VAP
3. Terminology
4. VAP Message Structure
5. VAP Transactions
    5.1. Transport and Connection Management
    5.2. Requestor Procedures
        5.2.1. Generating Requests
        5.2.2. Receiving Responses
    5.3. Responder Behaviors
        5.3.1. Receiving Requests
        5.3.2. Sending Responses
6. State Model
7. Protocol Versioning
8. ViPR Client Procedures
    8.1. Discovery
    8.2. Registration
    8.3. Unregistering
    8.4. Publishing Services
        8.4.1. VService
        8.4.2. ViPR Number Service
    8.5. Updating the VService
    8.6. Uploading VCRs
    8.7. Subscribing to Number Service
    8.8. Unsubscribing to Services
    8.9. Receiving Notify

8.10. Receiving PublishRevoke
9. ViPR Server Procedures
9.1. Connection Establishment
9.2. Registration
9.3. Unregistration
9.4. Publication
9.4.1. VService
9.4.2. ViPR Number Service
9.5. Unpublish
9.6. Subscribe
9.7. Unsubscribe
9.8. UploadVCR
9.8.1. Originating
9.8.2. Terminating
9.9. Sending Notify
9.10. Sending PublishRevoke
10. Syntax Details
10.1. XML Schema for VService
10.2. XML Schema for ValInfo
10.3. VAP Attributes
10.3.1. USERNAME
10.3.2. REALM
10.3.3. MESSAGE-INTEGRITY
10.3.4. ERROR-CODE
10.3.5. Client-Name
10.3.6. Client-Handle
10.3.7. Protocol-Version
10.3.8. Client-Label
10.3.9. Keepalive
10.3.10. ServiceIdentity
10.3.11. ServiceVersion
10.3.12. ServiceContent
10.3.13. SubscriptionID
10.3.14. CallDirection
10.3.15. StartTime
10.3.16. StopTime Attribute
10.3.17. CallingNum Attribute
10.3.18. CalledNum Attribute
10.3.19. Quota Attribute
10.3.20. DHTLifetime Attribute
11. Security Considerations
11.1. Outsider Attacks
11.2. Insider Attacks
12. IANA Considerations
13. References
13.1. Normative References
13.2. Informative References
Appendix A. Release notes
A.1. Modifications between rosenberg-03 and rosenberg-02

§ Authors' Addresses

1. Introduction to ViPR

TOC

[VIPR-OVERVIEW] (Rosenberg, J., Jennings, C., and M. Petit-Huguenin, “Verification Involving PSTN Reachability: Requirements and Architecture Overview,” October 2010.) introduces a new technology, called Verification Involving PSTN Reachability (ViPR), which enables VoIP federation between domains, over the Internet. ViPR is a hybrid technology that combines together the PSTN, P2P networks, and SIP. In doing so, it addresses the phone number routing problem and anti-spam problems that have been the most significant barriers to widespread deployment of SIP inter-domain federation.

It is assumed that readers of this document have read and understood [VIPR-OVERVIEW] (Rosenberg, J., Jennings, C., and M. Petit-Huguenin, “Verification Involving PSTN Reachability: Requirements and Architecture Overview,” October 2010.).

One of the key protocols used in ViPR is the ViPR Access Protocol (VAP). VAP connects call agents, such as phones, SBCs and IP PBXs, to a ViPR server. This document defines the VAP protocol in detail.

2. Overview of VAP

TOC

A high level view on the ViPR architecture is shown in Figure 1 (Architecture). This architecture is discussed in more detail in [VIPR-OVERVIEW] (Rosenberg, J., Jennings, C., and M. Petit-Huguenin, “Verification Involving PSTN Reachability: Requirements and Architecture Overview,” October 2010.).

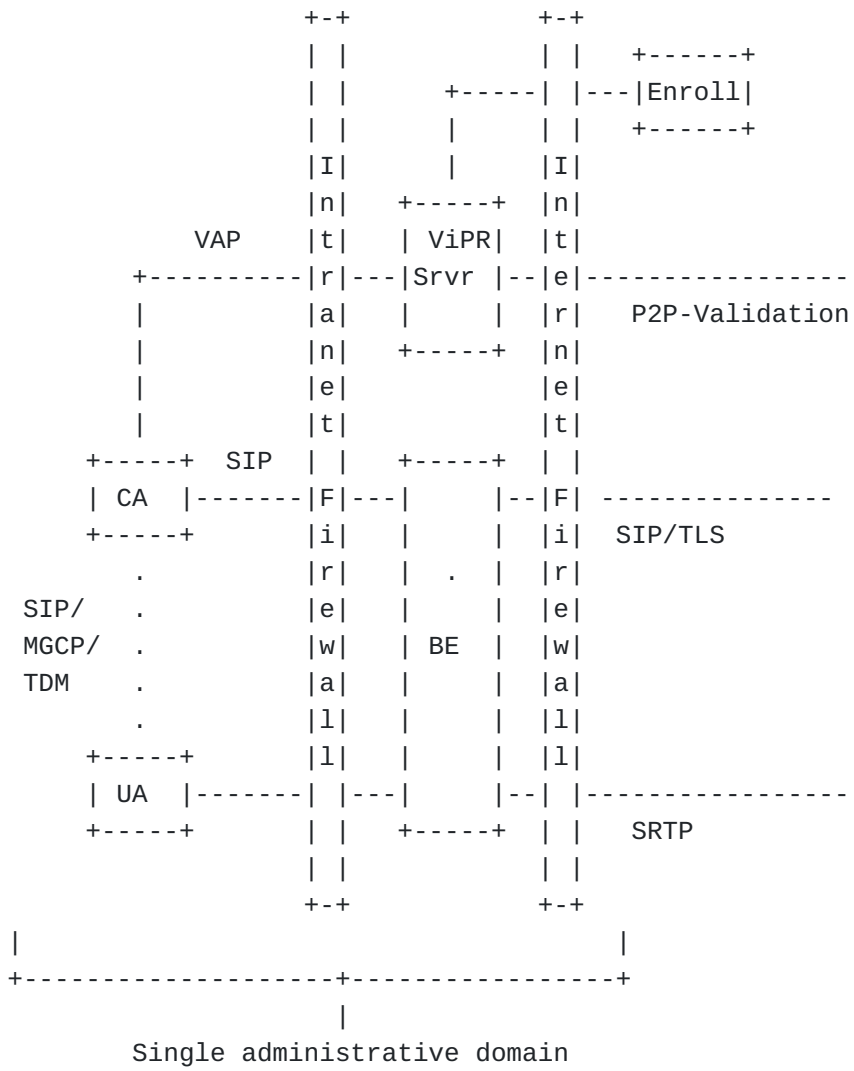

Figure 1: Architecture

A key component of this architecture is the ViPR server. The ViPR server is responsible for connecting to the P2P network, publishing phone numbers into that network, performing validation, and learning new routes. The ViPR server performs those functions on behalf of one or more call agents. This requires a protocol to run between the call agents and the ViPR server. This protocol is called VAP - the ViPR Access Protocol.

VAP is a client-server protocol that runs between the call agent and the ViPR server. VAP is a simple, binary based, request/response protocol. It utilizes the same syntactic structure and transaction state machinery as STUN [RFC5389] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for NAT (STUN),” October 2008.), but otherwise is totally distinct from it. VAP clients initiate TCP/TLS connections towards the ViPR server. The ViPR server

never opens connections towards the call agent. This allows the ViPR servers to run on the public side of NATs and firewalls.
Once the connections are established, the call agent sends a Register message to the ViPR server. This register message primarily provides authentication and connects the client to the ViPR server. VAP provides several messages for different purposes:

*Publish: The Publish message informs the ViPR server of service information. There are two types of Publishes supported in ViPR. The first is the ViPR Service (VService). This informs the ViPR server of the SIP URIs on the call agent and black and white lists used by the ViPR server to block validations. The ViPR server stores that information locally and uses it during the validation process, as described above. The second Publish is the ViPR Number Service. The ViPR server, upon receiving this message, performs a Store operation into the DHT.

*UploadVCR: This message comes in two flavors - an originating and terminating message. An originating UploadVCR comes from a call agent upon completion of a non-ViPR call to the PSTN. A terminating UploadVCR comes from an agent upon completion of a call received FROM the PSTN. The ViPR server behavior for both messages is very different. For originating UploadVCR, the ViPR server will store these, and at a random time later, query the DHT for the called number and attempt validation against the ViPR servers that are found. For a terminating UploadVCR, the ViPR server will store these, awaiting receipt of a validation against them.

*Subscribe: Call agents can subscribe for information from the ViPR server. There is one service that call agent can subscribe for: Number Service. When a new number is validated, the ViPR server will send a Notify to the call agent, containing the validated number, the ticket, and a set of SIP trunk URIs.

*Notify: The ViPR server sends this message to the call agent when it has an event to report for a particular subscription.

The VAP protocol provides authentication by including an integrity object in each message. This integrity message is the hash of the contents of the message and a shared secret between the ViPR server and the client. VAP can also be run over TLS, which enhances security further.

TOC

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.).

---

4. VAP Message Structure

TOC

VAP messages follow the syntax and structure of Session Traversal Utilities for NAT (STUN) [RFC5389] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)," October 2008.). It also shares the same transaction model as STUN. However, aside from its common syntax and transaction model, STUN and VAP are unrelated.
VAP messages are encoded in binary using network-oriented format (most significant byte or octet first, also commonly known as big-endian). The transmission order is described in detail in Appendix B of RFC791 (Postel, J., "Internet Protocol," September 1981.) [RFC0791]. Unless otherwise noted, numeric constants are in decimal (base 10).

All VAP messages MUST start with a 20-byte header followed by zero or more Attributes. The VAP header contains a VAP message type, message length, magic cookie and transaction ID.

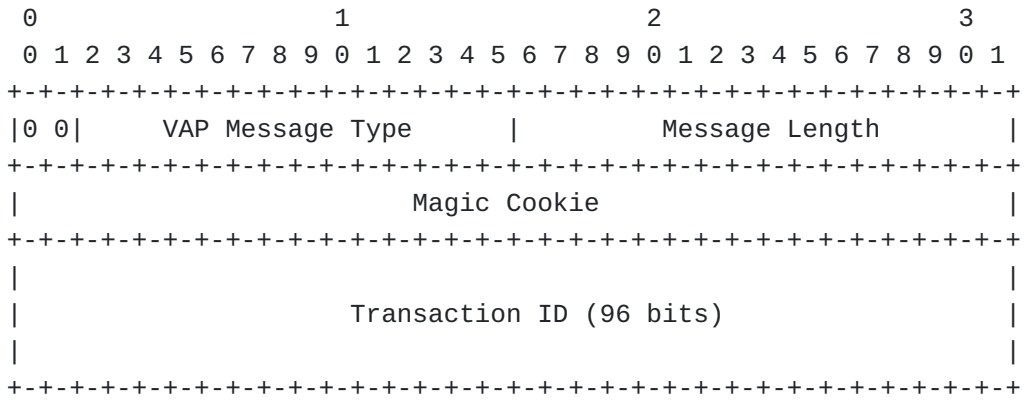

Figure 2: Format of VAP Message Header

The most significant two bits of every VAP message MUST be zeroes. The message type defines the message class (request, success response, failure response) and the message method (the primary function) of the

VAP message. Although there are four message classes, there is only one type of transaction in VAP: request/response transactions (which consist of a request message and a response message). Response classes are split into error and success responses to aid in quickly processing the VAP message.

The message type field is decomposed further into the following structure:

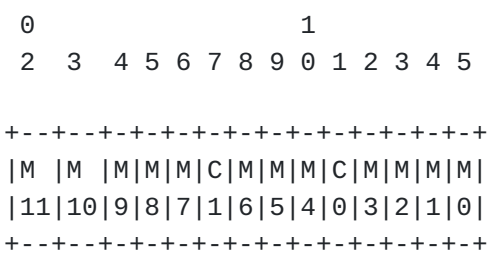

Figure 3: Format of VAP Message Type Field

Here the bits in the message type field are shown as most-significant (M11) through least-significant (M0). M11 through M0 represent a 12-bit encoding of the method. C1 and C0 represent a 2 bit encoding of the class. A class of 0b00 is a Request, a class of 0b10 is a success response, and a class of 0b11 is an error response. The method and class are orthogonal, so that for each method, a request, success response, error response and indication are defined for that method.
The magic cookie field MUST contain the fixed value 0x41666679 in network byte order (note that this is a different value than STUN).
The transaction ID is a 96 bit identifier, used to uniquely identify VAP transactions. For request/response transactions, the transaction ID is chosen by the VAP client for the request and echoed by the server in the response. The transaction ID MUST be uniformly and randomly chosen from the interval 0 .. 2**96-1, and SHOULD be cryptographically random. The client MUST choose a new transaction ID for new transactions. Success and error responses MUST carry the same transaction ID as their corresponding request.
The message length MUST contain the size, in bytes, of the message not including the 20 byte VAP header. Since all VAP attributes are padded to a multiple of four bytes, the last two bits of this field are always zero.
Following the VAP fixed portion of the header are zero or more attributes. Each attribute is TLV (type-length-value) encoded. The details of the attributes themselves is given in Section 10.3 (VAP Attributes).
The methods defined in VAP, and their corresponding method values, are:

Method	Value
Register	0x001
Unregister	0x002
Publish	0x004
Unpublish	0x005
PublishRevoke	0x006
Subscribe	0x007
Unsubscribe	0x008
Notify	0x00a
UploadVCR	0x00b

Figure 4: VAP Methods

After the VAP header are zero or more attributes. Each attribute is TLV encoded, with a 16 bit type, 16 bit length, and variable value. Each attribute MUST end on a 32 bit boundary:

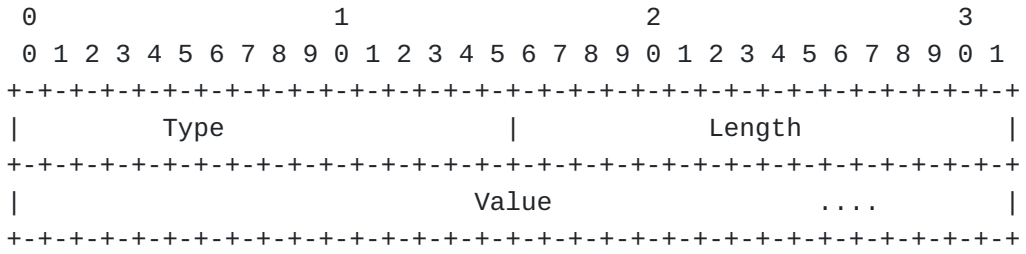

Figure 5: VAP Attributes

The Length refers to the length of the actual useful content of the Value portion of the attribute, measured in bytes. Since VAP aligns attributes on 32 bit boundaries, attributes whose content is not a multiple of 4 bytes are padded with 1, 2 or 3 bytes of padding so that they are a multiple of 4 bytes. Such padding is only needed with attributes that take freeform strings, such as USERNAME. For attributes that contain more structured data, the attributes are constructed to align on 32 bit boundaries. The value in the Length field refers to the length of the Value part of the attribute prior to padding - i.e., the useful content. Consequently, when parsing messages, implementations will need to round up the Length field to the nearest multiple of four in order to find the start of the next attribute.

5. VAP Transactions

TOC

This section describes the general behavior of VAP transactions, regardless of the method.

5.1. Transport and Connection Management

TOC

VAP runs only over TCP. UDP is not supported. As a consequence, transactions are simple. For each transaction, the client sends a single request, and the server sends a response.

VAP can also be run over TLS. The server MUST implement TLS, and the client SHOULD utilize it. The TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite MUST be implemented. The client MUST verify that the server certificate matches a configured value associated with the ViPR server that is to be used. The server MUST accept any certificate from the client. Client authentication is performed using a simple digest technique.

Reliability of VAP over TCP and TLS-over-TCP is handled by TCP itself, and there are no retransmissions at the VAP protocol level. However, for a request/response transaction, if the client has not received a response by Ti seconds after it sent the SYN to establish the connection, it considers the transaction to have timed out. Ti SHOULD be configurable and SHOULD have a default of 39.5s.

In addition, if the client is unable to establish the TCP connection, or the TCP connection is reset or fails before a response is received, any request/response transaction in progress is considered to have failed.

The client MAY send multiple transactions over a single TCP (or TLS-over-TCP) connection, and it MAY send another request before receiving a response to the previous. The client SHOULD keep the connection open until it

*has no further VAP requests to send over that connection, and;

*has no outstanding subscriptions

At the server end, the server SHOULD keep the connection open, and let the client close it, unless the server has determined that the connection has timed out (for example, due to the client disconnecting from the network). The server SHOULD NOT close a connection if a request was received over that connection for which a response was not sent. A server MUST NOT ever open a connection back towards the client in order to send a response. Servers SHOULD follow best practices regarding connection management in cases of overload.

5.2. Requestor Procedures

TOC

Though VAP is a client/server protocol, the ViPR server can asynchronously send requests towards the client call agent. As such, this section defines transaction rules in terms of the requestor (the entity sending the request) and the responder (the entity receiving the request).

5.2.1. Generating Requests

TOC

The requestor MUST construct a request message based on the syntax in Section 4 (VAP Message Structure). The message class MUST be a request. The method depends on the method of the request.
The requestor MUST add a MESSAGE-INTEGRITY, REALM and USERNAME attribute to the request message. The USERNAME contains a string which is the provisioned username identifying the client to the VAP server. The REALM attribute MUST have the value of "ViPR". The MESSAGE-INTEGRITY is computed as described in Section 10.3.3 (MESSAGE-INTEGRITY). That computation relies on a 16-byte key. The 16-byte key for MESSAGE-INTEGRITY HMAC is formed by taking the MD5 hash of the result of concatenating the following five fields: (1) The username, with any quotes and trailing nulls removed, (2) A single colon, (3) The realm, with any quotes and trailing nulls removed, (4) A single colon, and (5) The password, with any trailing nulls removed. Note that the password itself never appears in the message.
This format for the key was chosen so as to enable a common authentication database for SIP, which uses digest authentication as defined in RFC 2617 [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.).
The request will contain other attributes depending on the method.

5.2.2. Receiving Responses

TOC

All responses MUST first be authenticated by the requestor. Authentication is performed by first comparing the Transaction ID of the response to an outstanding request. If there is no match, the requestor MUST discard the response. Then the requestor SHOULD check the response for a MESSAGE-INTEGRITY attribute. If not present, it MUST discard the response, except for error responses with response codes

431 and 436. If MESSAGE-INTEGRITY is present, the requestor computes the HMAC over the response. The key that is used MUST be same as used to compute the MESSAGE-INTEGRITY attribute in the request.
If the computed HMAC matches the one from the response, processing continues. If the response was discarded, in cases where the failure is due to an implementation error, this will cause timeout of the transaction.
If the response is an Error Response, the requestor checks the response code from the ERROR-CODE attribute of the response. For a 400 (Bad Request) response code, the requestor SHOULD generate an alarm (a notification here refers to some kind of indication, sent to the administrator of the system, indicating an error condition. Notification mechanisms include SNMP alarms, logs, syslog, and so on, and are a matter of local implementation) containing the reason phrase. For a 431 (Integrity Check Failure) response code, this is typically caused by a mis-provisioning of the password. The requestor SHOULD generate an alarm and SHOULD NOT retry.
If the requestor receives a 436 (Unknown Username) response, it means that the username it provided in the request is unknown. This is typically due to a provisioning error, a consequence of a mismatched username. The requestor SHOULD generate an alarm.
The requestor MUST ignore any attributes from the response whose attribute type were not understood by the requestor.

5.3. Responder Behaviors

TOC

5.3.1. Receiving Requests

TOC

A responder will receive requests on an existing TCP connection, either one initiated by the client, or the one accepted by the ViPR server.
If a responder cannot process a request because the request does not meet the syntactic requirements necessary for the processing described below, the responder SHOULD reject the request with an error response and include an ERROR-CODE attribute with a response code of 400 (Bad Request). If the request is so malformed that a response cannot be generated, the request is just dropped. Error codes for specific failures are not provided, since these failures would not be seen in a functionally correct system. The protocol only provides error codes for errors that can arise due to misconfiguration or network error. Note, however, that a responder SHOULD NOT verify that a requestor has generated the request in full compliance to this specification; it should only validate what it needs to perform the processing described for handling the request.

First, the responder authenticates the request. The request will contain a USERNAME, REALM, and MESSAGE-INTEGRITY attribute. If the USERNAME is unknown, the responder generates an error response with an ERROR-CODE attribute with a response code of 436 (Unknown Username). The response MUST include the REALM, but MUST omit the MESSAGE-INTEGRITY attribute.
The responder computes the HMAC over the request. If the computed HMAC differs from the one from the MESSAGE-INTEGRITY attribute in the request, the responder MUST generate an error response with an ERROR-CODE attribute with a response code of 431 (Integrity Check Failure). This response MUST include a REALM but MUST omit the MESSAGE-INTEGRITY attribute.
The responder MUST ignore any attributes from the request whose attribute type were not understood by the responder.

5.3.2. Sending Responses

TOC

To construct the response the responder follows the message structure described in Section 4 (VAP Message Structure). The message type MUST indicate either a success response or error response class and MUST indicate the same method as the request. The responder MUST copy the transaction ID from the request to the response.
The attributes that get added to the response depend on the type of response.
When sending an error response, the server MUST add an ERROR-CODE attribute containing the error code. The reason phrase is not fixed, but SHOULD be something suitable for the error code.
All responses except for an error response with ERROR-CODE of 431 and 436 will contain a MESSAGE-INTEGRITY attribute. All responses will contain a REALM attribute. The computation of the message integrity is based on the same username value present in the request (along with its corresponding password); however the response SHOULD NOT contain the USERNAME attribute.
All responses MUST be sent on the same TCP connection on which the request was received. If this connection has closed, the responder MUST NOT open a new connection in order to try to send the response. The transaction is considered failed in this case.

---

6. State Model

TOC

The state model for VAP is shown in Figure Figure 6 (VAP State Model). This state is built up as a consequence of the primary messages which build state on the ViPR server: Register, Publish, UploadVCR and Subscribe.

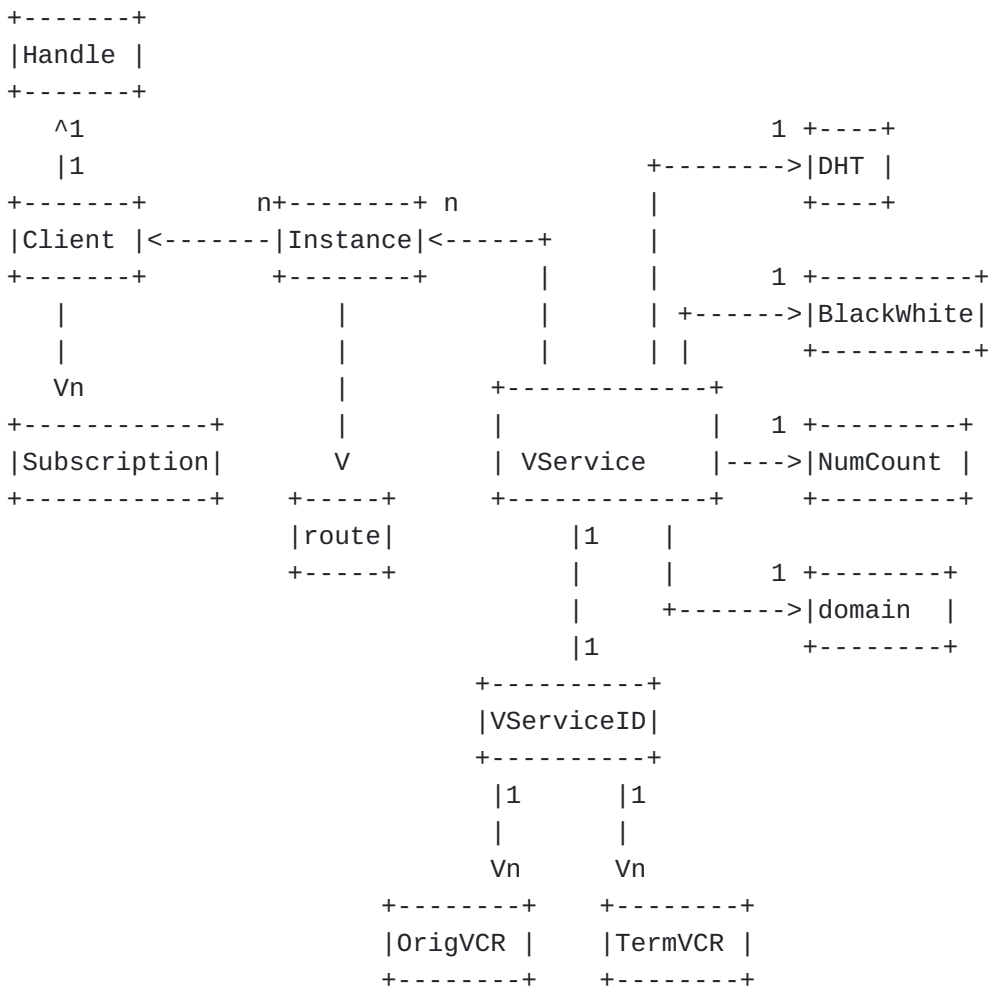

Figure 6: VAP State Model

It is important to understand that the ViPR client publishes two unique sets of information to the ViPR server:
1. The set of numbers that are reachable by the client through a particular ViPR service,
2. The set of ViPR services
Both of these are uploaded from the client to the ViPR server using a VAP Publish operation. The ViPR clients have the concept of a "ViPR Service" (not to be confused with ViPR server). A ViPR service is a unique instance of ViPR processing in a call agent - and is associated with a specific DHT, specific routes, specific domain, specific set of numbers to use, and specific set of policies governing operation. When a client publishes a number, it is always associated with a specific

ViPR service, or VService. Multiple clients can publish the same VServices, and they will differ only in the routes associated with that VService, as each client will have its own route to reach the same VService.

The ViPR server actively tracks the association of clients, VServices, routes, DHTs, BlackWhite lists, and VServiceIDs. Number publications and VService publications are differentiated from each other by different serviceID values in attributes in the Publish request. To be thoroughly confusing, this serviceID is not the same as a VServiceID. ServiceID refers to whether something is a VService publication or number publication, and is an enumerated value, whereas a VServiceID is an instance ID for a particular VService. The ViPR server only actually stores the VService publications; when receiving a Publish for a number service, the corresponding data is written directly to the DHT and then forgotten by the ViPR server. The ViPR server doesn't take any responsibility for removing the state or for keeping it fresh. All of this is the responsibility of the ViPR client. Consequently, VAP itself is not responsible for maintaining this information.

Firstly, when a client connects, it will Register to the ViPR server. That creates an instance of the client object, which is assigned a unique handle that identifies it. The client object is one of the key pieces of state (ViPR service being the other). All subsequent messaging from the client includes that Client-Handle, allowing the ViPR server to immediately determine the client associated with the messaging.

The client can issue subscriptions for services over its connection to the ViPR server. The ViPR server remembers the set of subscriptions from that client.

The VService publication builds the next large block of state. When a VService publication is received from a client, the ViPR server creates the VService object if it didn't have one yet for that VServiceID. Each distinct instance of a VService publication gets linked to it, and each distinct instance is, in turn, associated with one or more routes. Each route has a SIP URI, but the internal structure of the route is opaque to the ViPR server. It parses no deeper than the route element itself; the contents are not parsed, examined or checked by the ViPR server. This allows for future extensibility on how call routing is done. The VService itself has a numberCount, domain, BlackWhite list and DHT, all of which are learned from the VService publication. The VServiceID is 1-1 associated with each VService.

Finally, each UploadVCR, whether it is originating or terminating, contains a VServiceID as well. This binds it to a particular VService. It is important to note that, the linkage from VCRs to VServices is indirect, through the VServiceID. This allows a temporary outage to break all client connections, which will delete the VService objects, but keep the VCRs and the VServiceIDs. When the clients reconnect, the VServices are rebuilt, along with their IDs, and once again can be linked to the VCRs.

When the VAP connection terminates, the client object and subscription state from the corresponding client is destroyed. Any instances of a VService from that client are destroyed. If there are no longer any instances of the VService left, the VService itself is destroyed. The VCRs are not affected by the termination of a connection from a client. When the client TCP connection breaks or keepalives cease to be sent, the ViPR server will remove the registration, subscription and VServiceID to SIP trunk/DHT mappings. Similarly, on the client side, if the TCP connection breaks, the client must create a new TCP connection, register without a handle, subscribe and performs its VService publications.

The VAP state above is, in addition, utterly and completely orthogonal to the state of the DHT itself. That state is driven through number service publications, which cause storage operations into the DHT.

---

7. Protocol Versioning

TOC

Each version of VAP has a major and minor version number. This specification describes major version 1, minor version 0. It is anticipated that the protocol may require updating in the future.

If an update can be done such that an older client will work with a newer server, and an older server with a newer client, this MUST be done using an increase in the minor version number within the major version. This would typically include bug fixes and minor extensions.

If a protocol change is such that it cannot be understood by previous servers and clients, this MUST be done using an increase in the major version number of the protocol.

This specification further requires that, in addition to the most recent version of the protocol they understand, a client MUST understand the previous major version number. For example, a client supporting version 2.1 would also need to support version 1.0.

The protocol version number is included in client register messages, and negotiation as part of that exchange.

This allows for a graceful upgrade procedure. When a new version of the protocol is to be rolled out, the clients are upgraded first, each in turn. When they are upgraded, they'll come back, but during registration, notices that the servers only support a previous major version. The clients thus switch to the previous version of the protocol. Once all of the clients are updated, the servers can be updated. When the clients connect to them, they will utilize the newest version of the protocol.

TOC

8. ViPR Client Procedures

8.1. Discovery

TOC

VAP provides no discovery mechanism. The client must be provisioned with the domain names and/or IP addresses and ports of its ViPR servers. Typically, a client will be provisioned with two servers - a primary and a backup.

8.2. Registration

TOC

Once a TCP connection is established, the client MUST perform a registration. This applies to all TCP connections held by the client for purposes of high availability.

The client constructs a Register request based on the basic client procedures in Section 5.2 (Requestor Procedures). In addition, the client MUST include the Client-Name attribute. This field is used strictly for debugging purposes and indicates the name of the client to the server.

If the client is registering for the first time towards this ViPR server, the registration MUST omit the Client-Handle attribute.

If the client is registering for the first time towards this ViPR server (and thus there was not Client-Handle attribute), the client MUST include a Protocol-Version attribute in the request. This includes the major and minor version number of the most recent version of the protocol supported by the client. For purposes of extensibility, in addition to their current version of the client protocol, a client MUST support the previous major version as well.

The client MUST include the Client-Label attribute in the request. However, it is not used and its contents are arbitrary.

Once constructed, the client sends the Register request to the ViPR server. The response is processed using the general techniques in Section 5.2 (Requestor Procedures). Assuming a success response is ultimately received, it indicates that the client has successfully registered. This response will contain a Client-Handle attribute. The client MUST retain this handle and store it for the lifetime of the clients connection to the server. The response will also contain the Keepalive attribute, which tells the client how often it needs to keepalive its registration to the server.

If the response to the initial Register request (one without a Client-Handle) is an error response with an ERROR-CODE attribute with a response code of 478, it means that the server does not support the major protocol version signaled by the client. The client MUST extract

the Protocol-Version attribute from the error response. This attribute indicates the major and minor versions supported by the server. Based on the principles in Section 7 (Protocol Versioning), the client will be able to support a version of the protocol that has a major protocol version matching the one in the Protocol-Version attribute of the error response. The client MUST switch to this version of the protocol, and then MUST generate a new Register request (without a Client-Handle), indicating a Protocol-Version equal to the new, lower version of the protocol.

If the response to the initial Register request (one without a Client-Handle) is an error response with an ERROR-CODE attribute with a response code of 477, it means that the server believes that the client has already registered on this connection. There has been a state synchronization error. The client SHOULD generate an alarm, and then tear down the TCP connection. It MUST open a new TCP connection, and then generate a fresh Register request (without a Client-Handle) over that connection.

If the Register message was for an existing connection (and thus a keepalive), and thus included the Client-Handle attribute in the request, but the response was a Register Error response with an ERROR-RESPONSE with a response code of 471, the client MUST consider this a failure of the connection. It SHOULD attempt a new connection and a new Register, but without a Client-Handle.

During an initial Register (one that omits Client-Handle), the client MUST NOT generate any subsequent requests until that Register transaction completes.

If the TCP connection fails, the client needs to reconnect and create a new registration without the handle, and furthermore, resubscribe and republish as needed. In other words, on the client side, the lifetime of the handle is equal to the lifetime of the TCP connection. The server also holds onto the handle as long as the connection is active. However, it will also watch for refreshes of registrations, and if it doesn't see one fast enough, remove the client registration, the handle, and state received from that client, as well.

8.3. Unregistering

TOC

A Client that wishes to terminate its connection gracefully does so using the Unregister request. This request is first constructed as described in Section 5.2 (Requestor Procedures). Once constructed, the client MUST add the Client-Handle attribute to the request, and send it to the ViPR server.

If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

If the response was an error response with an ERROR-RESPONSE attribute with a response code of 471, the client MUST consider this a failure of the connection. It indicates a synchronization error between client and server. The client SHOULD generate an alarm.
If the response was an error response and was of type 474, it means that the client sent an Unregister request on a TCP connection but had not yet registered. If the client had registered, there has been some kind of synchronization error. The client SHOULD generate an alarm.
In all cases, success or error responses, the client MUST consider all subscriptions to this server terminated, and consider all published VServices to this server as unpublished. The client MUST terminate the TCP connection after the response has been received.

8.4. Publishing Services

TOC

Publish requests inform the ViPR server of information from the client. There are two types, VService publications and number publications. These differ in the value of the ServiceIdentity attribute.
All publications contain a ServiceContent attribute which contains an XML element that defines the service. The schema for the ServiceContent element depends on whether the publication is a VService or number publication.
The Publish request MUST contain a ServiceVersion attribute. This attribute is a version number that increments by at least one every time a particular service (identified by a unique VService, instance, service ID and sub-service ID value) changes in any way. If the service data different from the previous published value, the ServiceVersion attribute MUST increase. If the service data is the same as the previous published value, the ServiceVersion SHOULD stay the same, but MAY increase. Consequently, increasing version numbers are not a guarantee that there was a change; only that lack of increasing version number is a guarantee that there was no change.
If a client loses track of the previous version number of the service (due, for example, to a restart), it MUST choose a new instance ID and then it can reset the ServiceVersion.
Finally, the Publish Request MUST contain a ServiceContent attribute. This attribute contains the actual service data. Its actual structure and syntax are a function of the service and sub-service.
If the response was an error response and was of type 472, it means that the client didn't increment the sequence number. More likely, it indicates that the client has inadvertently forgotten the version number of the service and gotten out of sync with the server. The client SHOULD choose a new instance ID for this service, withdraw the old one, and publish the new one.
If the response was an error response and was of type 474, it means that the client sent a Publish request on a TCP connection but had not

yet registered. If the client hadn't registered, it MUST now do so. If it had registered, there has been some kind of synchronization error. The client SHOULD generate an alarm. Then, it MUST generate a new register (without the Client-Handle), flushing all subscriptions.
If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.
If the response was a success, the publication has been accepted.

8.4.1. VService

TOC

The VService indicates the critical information for the VService identified by the VService ID. Typically, a call agent will run on many servers, each of which is listening for SIP traffic on a specific IP address and port. Each such IP address and port forms a particular instance of the VService, and represents an alternative SIP destination for receiving incoming calls. The instance ID is a unique identifier, within the scope of the VServiceID, which identifies that call agent server.
The additional information placed into the VService publication will not vary amongst different instances. That information is:

- The DHT that the client wishes its numbers to be published into for this VService. This must always be the name of the public ViPR DHT, which is "Quetzalcoatl".

- The domain name associated with this VService, e.g., example.com. This domain name is used by the ViPR server at the end of the validation process.

- The set of routes which can be used to reach a SIP server on the call agent instance. Each route contains a SIP URI, in addition to extensions to allow for future advanced routing. This parameter of the VService data is instance specific.

- a black/white list of domains. These are used by the ViPR server during the validation protocol. The white list contains the set of domains that this domain wishes to only federate with. The black list contains the list of domains that this domain does not wish to federate with.

- A count of the number of phone numbers being published for this VService. This is used for quota management on the ViPR server.

Note that the VService does not contain phone numbers. VService information is not stored into the DHT by the ViPR server. It is stored locally on the ViPR server and used to support the validation protocol. Section 10.1 (XML Schema for VService) defines the XML schema for the object included in the Publish request.
The SIP URI is constructed as follows:

1. The scheme MUST be sip.

2. The user part MUST be an identifier which is unique to this agent and is identical for all instances of that call agent. For example, if a call agent consists of two servers for purposes availability, and either can be used, the user part will be identical in the SIP URI published by each server.

3. The domain part MUST be the domain associated with this call agent, and MUST match certificates that the domain can obtain.

4. There MUST be a port and it MUST be the port on which incoming SIP invites can be received.

5. There MUST be an maddr URI parameter, and it MUST contain the IP address or hostname of the instance of the call agent server.

6. The transport URI parameter MUST be present and MUST be TCP.

There will be one or more URI per each instance of the call agent. The IP address in the URI MUST be a publicly reachable one. If the call agent is to be reached through a border element, the IP address and port on the border element MUST be used here.
The use of the IP address in the maddr parameter allows the system to operate without DNS support.
An example document for a VService on the public DHT is:

```
<?xml version="1.0" encoding="UTF-8"?>
<service-description
xmlns="http://www.cisco.com/namespaces/saf-uc" id="has7gg"
xmlns:vt="http://www.cisco.com/namespaces/viprtrunk"
schemaVersion="1.0">
<tns:vservice xmlns:tns="http://www.cisco.com/namespaces/viprtrunk">
 <tns:DHTname>Quetzalcoatl</tns:DHTname>
 <tns:DIDCount>3670</tns:DIDCount>
 <tns:domain>example.com</tns:domain>
 <tns:whitelist>
   <tns:domain>example.com</tns:domain>
   <tns:domain>foo.edu</tns:domain>
 </tns:whitelist>
 <tns:route>
   <tns:SIPURI>
sip:17ahhs7zpaksux6z5==@example.com:2371;maddr=1.2.3.4;transport=tcp
   </tns:SIPURI>
 </tns:route>
 </tns:vservice>
</service-description>
```

Figure 7: Example ServiceContent

The ViPR client SHOULD publish each ViPR trunk service to both its primary and backup ViPR server, for purposes of HA.

8.4.2. ViPR Number Service

TOC

The ViPR number service is used to publish the numbers that are associated with the VService. It is published as a separate service due to the differing state requirements associated with the numbers. For the VService, the ViPR server stores the information and does not actually publish it into the DHT. For ViPR number service, the ViPR server immediately writes the data into the DHT and doesn't actually store it locally. The ViPR server does not refresh the data in the DHT on its own, nor does it withdraw the data from the DHT when the client disconnects. The ViPR client is responsible for refreshing the data in the DHT by periodically refreshing each of its numbers in each DHT. The numbers in the DHT have a configurable expiration. Consequently, the ViPR client has to refresh the data prior to the expiration. There is no way in VAP to remove a number from the DHT; it is merely left to expire.

The ViPR client SHOULD publish each service to both its primary and backup ViPR server, for purposes of HA. Next, the client constructs a ViPR number service advertisement. Unlike VService advertisements, which utilize an XML object in the ServiceContent attribute, number services utilize only VAP attributes. The Publish message will contain a ServiceIdentity attribute and a CalledNum attribute. The VServiceID of the ServiceIdentity attribute indicates the VService for this number, and is used by the ViPR server to determine which DHT to publish into. The CalledNum attribute contains the number to be published into the DHT. The ServiceVersion attribute is not present.

8.5. Updating the VService

TOC

A client can change the VService information at any time. Typically, changes in the black or white list will require an updated VService publication, as will changes in the set of servers listening for incoming SIP traffic.

To update a VService, the client modifies its service description, and creates a new Publish request. This request is first formed as described in Section 4 (VAP Message Structure). This request MUST contain the ServiceIdentity attribute, identifying the service to be modified. The request MUST also contain the ServiceContent attributes, containing the relevant information for the service.

The request MUST contain a ServiceVersion attribute. That version number MUST be at least one higher than the version number in the previous publication for the same service (as identified by service ID, subservice ID and instance).

If the response was an error response and was of type 472, it means that the client didn't increment the sequence number. More likely, it indicates that the client has inadvertently forgotten the version number of the service and gotten out of sync with the server. The client SHOULD choose a new instance ID for this service, unregister, reconnect, re-register, and republish.

If the response was an error response and was of type 474, it means that the client sent a Publish request on a TCP connection but had not yet registered. If the client hadn't registered, it MUST now do so. If it had registered, there has been some kind of synchronization error. The client SHOULD generate an alarm. Then, it MUST generate a new register (without the Client-Handle).

If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

If a client is no longer capable of receiving SIP requests at the URI it previously published, it should remove its VService by sending an Unpublish request.

8.6. Uploading VCRs

TOC

When the call agent initiates or receives a call that goes towards the PSTN, whether it be through a PSTN gateway or through a SIP trunk to a service provider, the call agent MUST send an UploadVCR request to its primary server ViPR server. It SHOULD send its terminating UploadVCRs to its secondary ViPR server, and SHOULD NOT send its originating UploadVCRs to its secondary. The UploadVCR request is first constructed like any other VAP request. This means it will contain the USERNAME, REALM, and MESSAGE-INTEGRITY attributes.

In addition, it MUST contain a CallingNum, CalledNum, StartTime and StopTime attribute. The CallDirection attribute is set as described in Section 10.3.14 (CallDirection).

The UploadVCR request MUST contain a ServiceIdentity attribute. The serviceID is 100, the subservice ID is 3 (ViPR number service) and the VService ID must identify the VService for which this UploadVCR is associated. The instance is arbitrary and are ignored by the ViPR server.

If the response was an error response and was of type 474, it means that the client sent a UploadVCR request on a TCP connection but had not yet registered and had not yet sent a VService publication with a VServiceID matching that of the UploadVCR. If the client hadn't registered and published a matching VService, it MUST now do so. If it had, there has been some kind of synchronization error. The client SHOULD generate an alarm. Then, it MUST disconnect, generate a new register (without the Client-Handle) and a new VService publication.

If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

8.7. Subscribing to Number Service

TOC

In order to learn about validated numbers, a ViPR client MUST subscribe for the ViPR Number Service. The client should subscribe to just its primary ViPR server.

To create a subscription, the client creates a Subscribe request. The request is formed as described in Section 4 (VAP Message Structure). It MUST NOT be sent if the client has not previously generated a successful Register request on this connection.

Each initial Subscribe request MUST omit the SubscriptionID attribute; that attribute is only used when withdrawing a subscription. The client MUST include a ServiceIdentity attribute in the request. The service ID

MUST be 101, the subserviceID MUST be 3, the VServiceID MUST be the VServiceID for the VService from which learned numbers are desired, and the instance value MUST be all ones. This will cause the client to receive notifications upon validated numbers learned as a consequence of an UploadVCR for that VService.

8.8. Unsubscribing to Services

TOC

A client MAY terminate a subscription at any time. To do that, it sends an Unsubscribe request. This request MUST contain the SubscriptionID attribute identifying the subscription to be terminated. Note that this unsubscription will affect only the subscription identified by the subscription ID. Other subscriptions will continue to be in effect.
The client MAY generate additional Unsubscribe requests while the transactions for previous Subscribe, Publish or Unpublish requests are in progress. By definition a client can only Unsubscribe a subscription for which it had already received a successful response to a Subscribe request that created the subscription.
If the response was an error response and was of type 474, it means that the client sent a Subscribe request on a TCP connection but had not yet registered. If the client hadn't registered, it MUST now do so. If it had registered, there has been some kind of synchronization error. The client SHOULD generate an alarm. Then, it MUST generate a new register (without the Client-Handle).
If the response was an error response and was of type 476, it means that the client sent an Unsubscribe request for a subscription which does not exist. The client SHOULD generate an alarm, since a synchronization error has occurred. It should however proceed as if the withdrawal was successful.
If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

8.9. Receiving Notify

TOC

The ViPR server will generate a Notify request when a new number and route are learned. It will send this Notify request to all clients which have subscribed to the corresponding VService.
Once the client has received a successful response to its Subscribe request, the client MUST be prepared to receive Notify requests on the TCP connection to its ViPR server. When the client receives a Notify request, it searches for the SubscriptionID attribute in the request. This informs the client of the subscription that this notification is

associated with. If this subscriptionID is known to the client, it proceeds. Otherwise, it MUST generate a Notify error response with a 476 response code in an ERROR-RESPONSE attribute. When this occurs, there has been a synchronization error between the client and server in the set of valid subscriptions. This event SHOULD be alarmed, and the contents of the Notify not used.

The Notify request will contain a ServiceIdentity attribute and a ServiceContent attribute, in addition to the standard authentication attributes and the SubscriptionID attribute. The ViPR client must verify that the ServiceIdentity has service 100, subservice 3. It looks at the instance value, and checks that the topmost 64 bits of the instance contain a VServiceID that matches one for which the ViPR client is currently interested in learning about. The ViPR client then extracts the contents of the ServiceContent attribute. This will be an XML object, formatted as described below.

The client SHOULD store the phone number, SIP URI and Ticket. When receiving a future call to that phone number, it SHOULD send a SIP INVITE request to the SIP URI and include the ticket in an X-Cisco-ViPR-Ticket header field.

8.10. Receiving PublishRevoke

TOC

The PublishRevoke method is defined only for the VService, not for the Number Service. The ViPR server will send a PublishRevoke for a VService if the corresponding DHT is no longer available. The request will contain the ServiceIdentity attribute, which indicates the specific VService and instance that are being withdrawn. If these correspond to a known VService, the client should consider that service deactivated, and periodically try to republish it.

---

9. ViPR Server Procedures

TOC

9.1. Connection Establishment

TOC

The ViPR server MUST be prepared to receive incoming TCP or TLS connections on a configure port. Whether or not TCP or TLS is used, is a configured property of that port.

9.2. Registration

TOC

The purpose of registrations is to create VAP client objects, which represent a VAP connection and contain the state described in Section 6 (State Model), and then link those with a TCP connection. Each VAP connection can be considered a client object, linked to one and only one TCP connection at a time.

The first request that the server will receive over the TCP connection will be a Register request. This request is first processed as described in Section 5.3 (Responder Behaviors). Assuming those procedures succeed, the server checks for the Client-Handle attribute in the Register request. If present, the server checks if it currently has a client state object with that handle. If the client object was already bound to another TCP connection, that other TCP connection MUST be closed by the server, and then the new TCP connection MUST be bound to the client object.

If the Register request had a Client-Handle attribute, but there were no client objects with that handle, the server MUST generate an error response and MUST include an ERROR-CODE attribute with a response code of 471. This is due to a state synchronization error between the client and server. The server SHOULD generate an alarm.

If the Register did not have a Client-Handle attribute, it is a request to create a client object. The server examines the Protocol-Version attribute from the request. If the major version indicated in the attribute is higher than the version supported by the server, the server MUST reject the Register request with an error response and include an ERROR-CODE attribute with a response code of 478. That error response MUST include a Protocol-Version attribute that contains the major and minor protocol versions supported by the server.

Next, the server MUST create a new client object, and allocate a new Client-Handle for it. The Client-Handle MUST be unique amongst all other Client-Handles known to this server, across all clients that are connected to it.

If the registration succeeds, the server sends a success response. This response MUST include the Client-Handle attribute containing the handle created by the server. The response MUST include a Keepalive attribute, indicating the time in milliseconds that the server will need to see traffic from the client in order to continue to maintain the client object.

9.3. Unregistration

TOC

The client can gracefully disconnect by using an Unregister request. If the server receives an Unregister request on a TCP connection, it first looks for the client object bound to that connection. If there is

no client object bound to it, it means that the client has sent an Unregister request prior to registering, or there has been some kind of synchronization error. The server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474. Otherwise, if the client object is known to the server, it MUST generate a success response. Once it does, the server MUST destroy the client, its associated subscriptions, and published VService instances. It then sets a timer equal to thirty seconds. If the client has not closed the TCP connection bound to this client object, the server MUST close the TCP connection.
If, as a consequence of the deletion of those VService instances, there are no longer any instances left for a VService, that VService and its associated data (BlackWhite, DHT, numberCount) are removed.
Note that unregistration does not ever remove VCRs.

9.4. Publication

TOC

Behavior depends on whether the publication is for the VService or the ViPR number service.
The ViPR server extracts the ServiceIdentity attribute. If the value is not one of the following:

1. ServiceID is 101 and SubserviceID is 3.

2. ServiceID is 101 and SubserviceID is 4

the ViPR server sends a 400 response.

9.4.1. VService

TOC

If the Publish request is for service 100, sub-service 4, it indicates that this was for the VService. The ViPR server first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent a Publish request prior to registering, or there has been some kind of synchronization error. The server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.
The ViPR server extracts the contents of the ServiceContent attribute. This will be an XML object structured as defined in Section 10.1 (XML Schema for VService). It also extracts the VServiceID and Instance values from the ServiceIdentity attribute.
First, the ViPR server checks if it has any VService objects with the VServiceID from the publish.

*If it does, it replaces the BlackWhite, numberCount, domain, and DHTName parameters of that VService with the ones from the publish. Next, it checks to see if the instance is currently an instance associated with that VService:

* -If it is, the route elements for that instance are replaced with the route values from the publish.

-If it is not, a new instance object is created, associated with the client and the VService, and is linked with the route values from the publish.

*If it does not, it creates a new VService object, and associates it with the values of the BlackWhite, numberCount, domain, and DHTName parameters of the VService. Next, it creates a new instance, associates it with the VService, The route values from the publish are associated with that instance.

ViPR server sends a Publish success response. The ViPR server looks for all other ViPR services in the same DHT as the one from this Publish, it sums up their numberCounts, and includes that value in the "current" field of the Quota attribute in the Publish response. Since there is a limit on the count of the numbers that can be published into the DHT, this mechanism allows the ViPR server to inform the clients about the total usage across all clients of this ViPR server. Note further, that since the ViPR server itself does not have local memory of the numbers it stored into the DHT, the ViPR server cannot determine how many numbers have been placed into the DHT for a particular VService. That information is known only to the client. That is why the client informs the ViPR server of how many numbers it has published as part of the VService publication.
The ViPR server places its configured per-DHT limit for that DHT into the "limit" field in the Quota attribute in the Publish response. This tells the clients the maximum count of phone numbers which can be published.
The ViPR server includes a DHTLifetime attribute in the response. This attribute indicates the amount of time that data will remain in the DHT prior to be expunged. This is a configured property of the DHT.

9.4.2. ViPR Number Service

TOC

If the server receives a Publish request for service 100, sub-service 3, it indicates that this was for the ViPR Number Service. The ViPR server first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent a Publish request prior to registering, or there has been some

kind of synchronization error. The ViPR server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474. The ViPR server extracts the VServiceID from the ServiceIdentity attribute. It checks that, for that VServiceID, there is a VService object currently being stored. If not, the ViPR server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.

Next, the ViPR server extracts the number from the CalledNum attribute. The ViPR server extracts the DHT from the VService object associated with the VServiceID from the Publish. For the number, the ViPR server takes the number and treats it as an ASCII string, called the suffix seed.

Next, the ViPR server generates two additional strings. The first is formed by taking the suffix seed, and prepending the string "COPY1". The second is formed by taking the suffix seed and prepending the string "COPY2".

Each of the three values is passed through the SHA-1 hash function, producing 160 bits. The least significant 128 bits of this are taken. Those 128 bits, for each of the three values, form the Resource-ID against which a STORE is to be performed. Three separate stores are performed in order to provide security in the DHT. Each store operation writes an object into the DHT whose value is a dictionary (or map) entry.

Conceptually:

```
Store(Resource-ID, object)
```

Where Resource-ID is the 128 bit Resource-ID computed above. The stored object is a dictionary entry which has a key and a value:

```
Object = {key,value}
```

Here, the key is formed by taking the peerID of the storing node in hex format, without the "0x", appending a "+", followed by the VServiceID in hex format, without the "0x". For example, if a peer with peerID 0x8e60f5fab753037f64ab6c53947fd532 receives a Publish with a VServiceID of 0x7eeb6a7036478351, the resulting key is:

```
8e60f5fab753037f64ab6c53947fd532+7eeb6a7036478351
```

Both parts of this key are important. Using the peerID of the node performing the store basically segments the keyspace of the dictionary so that no two peers ever store using the same key. Indeed, the responsible node will verify the signature over the stored data and check the peerID against the value of the key, to make sure that a conflict does not take place. The usage of the VService allows for a single ViPR server to service multiple call agents, and to ensure that numbers published by one call agent (using one VServiceID) do not clobber or step on numbers published by another call agent (using a

different VServiceID). The responsible node does not verify or check the VServiceID.
In this version of the protocol, only one of the three stored objects is read. Three are stored to allow an enhancement in the future, which will read all three and use a simple voting algorithm to handle inconsistencies in the results. In this way, if a malicious node returns no result or fakes the result, as long as the remaining two results are retrieved, the validation process can continue. This means that the compromise of a single node has, with only extremely low probability (order Log(N)/N where N is the number of nodes in the ring) of being able to disrupt validation against a number.
The value of the dictionary entry is a sequence of TLV attributes, with the same format used by VAP. In this case, it is a single attribute, the peerID attribute. This attribute is populated with the peerID of the ViPR server in the DHT into which the STORE is being performed. The reason for using the TLV construct is to provide extensibility in the contents of the DHT. In the future, if needed, new ViPR nodes can add additional data, each with a specific attribute type. Older nodes will ignore any unknown attributes and go right for the peerID attribute, while newer ones can process the new and old attributes.
The Store operations are paced into the DHT at a fixed rate. The ViPR server maintains a queue. This queue is filled with store requests. The ViPR server services that queue at a fixed, provisioned rate, the Store Rate Limit. When serviced, the next Store operation in the queue is serviced. Because transactions from clients are pipelined, there can only be as many Store operations in the queue as there are simultaneously connected clients, times three (three Stores per Publish, one Publish at a timer per client). The Publish is then responded to with a success response. Note that, a success response is not sent until all three Store operations have been performed. If there is a failure due to inability to store into the DHT, the server returns a 481 error response. Note that a ViPR server cannot disambiguate the first Publish for a service and an updated Publish. It performs identical processing for each.
Note further that, the DHT itself will replicate each of the three stored values, producing a total of nine copies of each number into the DHT.

9.5. Unpublish

TOC

The ViPR client can only Unpublish for the VService.
The ViPR server extracts the VServiceID and instance from the ServiceIdentity in the Unpublish. It checks to see if there is an instance with that ID associated with the VService with that VServiceID. If there is, it removes the instance object and its associated SIPURI. If, as a consequence, there are no longer any

instances associated with the VService, it deletes the VService object and its associated attributes.

9.6. Subscribe

TOC

If the server receives a Subscribe request on a connection, it first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent a Subscribe request prior to registering, or there has been some kind of synchronization error. The server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.
The ViPR server checks that the ServiceIdentity from the request. If verifies that the ServiceID is 101 and the SubServiceID is 3. Any other combination causes the server to return a 400 response. The subscription is to the VServiceID identified in the ServiceIdentity attribute.
If the ServiceIdentity is valid, the server MUST create a new subscription object. It MUST allocate a SubscriptionID for this subscription. This ID MUST be unique across all SubscriptionIDs associated with this client. The subscription MUST be linked with the client object. It is not permitted for there to be multiple subscriptions from a client with identical VServices since each subscription is for a unique service/subservice/VServiceID/instance, the ViPR server can hash these to get a 32 bit SubscriptionID, or assign them sequentially and store the associations.
The ViPR server then checks the VServiceID from the ServiceIdentity attribute. The ViPR server adds a subscription object to the client object, and associates it with a SubscriptionID and the VServiceID which is being watched.
The server then generates a success response to the Subscribe request. It MUST include the SubscriptionID attribute in the response, identifying this subscription.

9.7. Unsubscribe

TOC

If the server receives an Unsubscribe request on a connection, it first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent an Unsubscribe request prior to registering, or there has been some kind of synchronization error. The server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.
Next, the server extracts the SubscriptionID attribute from the request. If it contains a SubscriptionID not known to the server, there

has been a synchronization error. The server MUST reject the Unsubscribe request with an error response and MUST include an ERROR-CODE attribute with a value of 476.
Assuming the SubscriptionID is known, the server MUST remove the subscription object from the client object, and destroy it. The server will therefore no longer send notifications associated with this subscription. The server MUST respond to the Unsubscribe request with a success response.

9.8. UploadVCR

TOC

The ViPR server first processes the request like any other VAP request, specifically it will perform the message integrity check and follow associated procedures.
If the UploadVCR was received on a TCP connection but the client had not yet registered over that connection, it is an error and the ViPR server returns a 474. If the client had registered, but the VServiceID from the ServiceIdentity doesn't match a known VService, the UploadVCR is rejected with a 474.
Otherwise, the ViPR server extracts the CallDirection, StartTime, StopTime, CallingNum and CalledNum attributes, and stores them. Further processing depends on whether it was an originating or terminating UploadVCR.

9.8.1. Originating

TOC

Once stored, the ViPR server starts timer Tv. Tv is selected as a random number, in seconds, starting from 30 and ending at the maximum validation time, which is a configured parameter of the ViPR Server for the DHT associated with the VService. The validation request - which includes the VCR - is stored until that timer fires. The validation request includes the details from the UploadVCR (calling, called numbers, start and stop time), along with the VService associated with the UploadVCR.
When the timer fires, the ViPR server examines the called party number. This number will be a plus followed by N digits. Using this number, it forms a lookup key K. K is equal to the least significant 128 bits of the SHA1 hash of the called party number in string form, including the + sign. Next, the ViPR server extracts VService associated with the VCR. It checks to see if this VService is currently being published. If so, it performs a lookup into the DHT using key K. Each DHT node has a queue on read transactions. These lookups are queued because the node has, per-DHT, a limit on the rate at which it will perform read requests.

Once the lookup request comes to the top of the queue and it can be serviced, the resulting fetch will be a result, a no-match, or a timeout. If there is a no-match or timeout, ViPR server processing is complete.
If there is a result, the ViPR server will now have all of the dictionary entries associated with the Resource-ID. Each dictionary entry is a key and a value. The key is the concatenation of a peerID and VServiceID, and the value is a set of TLV attributes. The ViPR server parses each dictionary entry as a sequence of TLV attributes, and extracts the first TLV value whose type is peerID (type 0x2008). From this, the ViPR server obtains a set of {peerID, VServiceID}s.
The ViPR server SHOULD perform validation, using the validation protocol [VIPR-PVP] (Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "The Public Switched Telephone Network (PSTN) Validation Protocol (PVP)," October 2010.). A ViPR server MAY use any algorithm of its choosing to determine whether a number should be validated once, many times, or not at all. When the ViPR server is satisfied that a number has been sufficiently validated, it SHOULD send a Notify.
Furthermore, during validation, the ViPR server SHOULD compare the domain of the learned number with the blacklist for the VService associated with the matching VCR. If the domain is on the blacklist or not on the whitelist, a Notify SHOULD NOT be sent.
If a Notify is to be sent as a consequence of a validation success, the ViPR server looks to see if there is currently a subscription from a client whose VServiceID matches the one from the VCR that triggered the validation that is causing the notification. For each matching one, it sends a Notify message. The ServiceContent in the Notify contains a ValInfo XML containing the SIPURI and ticket learned from the validation. It also contains the full E.164 number of the called number which validated.

9.8.2. Terminating

TOC

When the ViPR server receives a terminating UploadVCR, it stores the information, awaiting the receipt of a validation query. This information MUST be stored for a minimum whose value is a configured property of the DHT.

9.9. Sending Notify

TOC

The ViPR server MUST NOT send a Notify until it had already sent a response to the Subscribe message that created the subscription, for which the Notify is being sent.

When a Notify is to be sent, it must contain the SubscriptionID attribute associated with the subscription on which the notification is being sent. This will differ for each client that is subscribed. The Notify MUST contain the ServiceIdentity attribute, containing service 100, subservice 3, a VServiceID for the VService on which the number was learned, and an instance ID whose instance is all ones. The content of the ServiceContent attribute is an XML document, which is the scrubbed document from the ValExchange response. An example document is:

```
<?xml version="1.0" encoding="utf-8"?>
<valinfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="valinfo.xsd">
 <number>+17325552496</number>
 <ticket>7hasd88a7sd6a6d7989xkk8g7a6sdq78ekaz</ticket>
 <route>
   <SIPURI>
     sip:17ahhs$7zpaksux6z5==@example.com:2371;maddr=1.2.3.4
   </SIPURI>
 </route>
 <route>
   <SIPURI>
     sip:17ahhs$7zpaksux6z5==@example.com:2371;maddr=1.2.3.5
   </SIPURI>
 </route>
</valinfo>
```

Figure 8: Example Notify XML

9.10. Sending PublishRevoke

TOC

The ViPR server is only permitted to PublishRevoke the VService; it cannot withdraw Number Service publications. It should PublishRevoke published VServices when the corresponding DHT is no longer available. If this should happen, the ViPR server sends a PublishRevoke for each VService that was published which utilized the DHT which is no longer available. That PublishRevoke MUST include a ServiceIdentity attribute indicating the VServiceID and instanceID of the PublishRevoke service. Furthermore, it SHOULD include a ServiceContent attribute with the corresponding service description; this is used strictly for diagnostic

purposes and is not needed by the client. Once sent, the ViPR server removes that instance of that VServiceID from its internal state.

---

10. Syntax Details

TOC

10.1. XML Schema for VService

TOC

This document is included in publications for the ViPR service. Note its target namespace.

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema xmlns="http://www.cisco.com/namespaces/saf-uc"
 attributeFormDefault="unqualified" elementFormDefault="qualified"
 targetNamespace="http://www.cisco.com/namespaces/saf-uc"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="service-description">
   <xs:complexType>
     <xs:choice>
<xs:element name="vservice">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="DHTname" type="xs:string" />
      <xs:element name="DIDCount" type="xs:integer" />
      <xs:element minOccurs="1" maxOccurs="1" name="domain"
        type="xs:string" />
      <xs:choice minOccurs="0" maxOccurs="1">
        <xs:element
          xmlns:q1="http://www.cisco.com/namespaces/viprtrunk"
          name="blacklist" type="q1:whiteOrBlackList" />
        <xs:element
          xmlns:q2="http://www.cisco.com/namespaces/viprtrunk"
          name="whitelist" type="q2:whiteOrBlackList" />
      </xs:choice>
      <xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs:element
          xmlns:q1="http://www.cisco.com/namespaces/viprtrunk"
          name="route" type="q1:routeType" />
      </xs:sequence>
      <xs:any minOccurs="0" maxOccurs="unbounded"
        namespace="##other"
        processContents="lax" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
     </xs:choice>
     <xs:attribute name="schemaVersion" type="xs:string"
       use="required" />
     <xs:attribute name="id" type="xs:string" use="required" />
   </xs:complexType>
 </xs:element>
<xs:complexType name="whiteOrBlackList">
   <xs:sequence minOccurs="1" maxOccurs="unbounded">
     <xs:element name="domain" type="xs:string" />
   </xs:sequence>
 </xs:complexType>
 <xs:complexType name="routeType">
   <xs:sequence>
```

```
      <xs:element minOccurs="1" maxOccurs="unbounded" name="SIPURI"
        type="xs:string" />
      <xs:any minOccurs="0" maxOccurs="unbounded"
        namespace="##other" />
    </xs:sequence>
  </xs:complexType>
 </xs:schema>
```

Figure 9: VService XML Schema

10.2. XML Schema for ValInfo

TOC

This document is passed from the terminating ViPR server to the originating, containing the ticket, routes and number which was validated. The originating ViPR server verifies this and passes it to the client in VService notifications.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="valinfo">
   <xs:complexType>
     <xs:sequence minOccurs="0" maxOccurs="unbounded">
       <xs:element minOccurs="1" maxOccurs="1" name="number"
         type="xs:string" />
       <xs:element minOccurs="1" maxOccurs="1" name="ticket"
         type="xs:string" />
       <xs:element minOccurs="1" maxOccurs="unbounded" name="route"
         type="routeType" />
       <xs:any minOccurs="0" />
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:complexType name="routeType">
   <xs:sequence>
     <xs:element minOccurs="1" maxOccurs="unbounded" name="SIPURI"
       type="xs:string" />
     <xs:any minOccurs="0" maxOccurs="unbounded"
       namespace="##other" />
   </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Figure 10: ValInfo XML Schema

10.3. VAP Attributes

TOC

This section enumerates the attributes used by VAP. The attribute names and corresponding types are:

Attribute Name	Type
USERNAME	0x0006
MESSAGE-INTEGRITY	0x0008
REALM	0x0014
ERROR-CODE	0x0009
Client-Name	0x1001
Client-Handle	0x1002
Protocol-Version	0x1003
Client-Label	0x1005
Keepalive	0x1006
ServiceIdentity	0x1007
ServiceVersion	0x100b
ServiceContent	0x100c
SubscriptionID	0x100e
CallDirection	0x2001
StartTime	0x2002
StopTime	0x2003
CallingNum	0x2004
CalledNum	0x2005
peerID	0x2008
Quota	0x200a
DHTLifetime	0x200b

Figure 11: VAP Attributes

10.3.1. USERNAME

TOC

The USERNAME attribute is used for authentication. It identifies the shared secret used in the message integrity check. Consequently, the USERNAME MUST be included in any request that contains the MESSAGE-INTEGRITY attribute.
The value of USERNAME is a variable length opaque value of UTF-8 characters. Note that, as described above, if the USERNAME is not a multiple of four bytes it is padded for encoding into the VAP message, in which case the attribute length represents the length of the USERNAME prior to padding.

TOC

10.3.2. REALM

The REALM attribute is present in requests and responses. It contains text which meets the grammar for "realm" as described in RFC 3261 [RFC3261] (Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol,” June 2002.), and will thus contain a quoted string (including the quotes).
The value of this attribute MUST always be "ViPR".

10.3.3. MESSAGE-INTEGRITY

TOC

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 [RFC2104] (Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” February 1997.) of the message. The MESSAGE-INTEGRITY attribute can be present in any message type. Since it uses the SHA1 hash, the HMAC will be 20 bytes. The text used as input to HMAC is the message, including the header, up to and including the attribute preceding the MESSAGE-INTEGRITY attribute. That text is then padded with zeroes so as to be a multiple of 64 bytes. The MESSAGE-INTEGRITY attribute MUST be the last attribute in the message.
The 16-byte key for MESSAGE-INTEGRITY HMAC is formed by taking the MD5 hash of the result of concatenating the following five fields: (1) The username, with any quotes and trailing nulls removed, (2) A single colon, (3) The realm, with any quotes and trailing nulls removed, (4) A single colon, and (5) The password, with any trailing nulls removed. Note that the password itself never appears in the message.
Since the hash is computed over the entire message, it includes the length field from the message header. This length indicates the length of the entire message, including the MESSAGE-INTEGRITY attribute itself. Consequently, the MESSAGE-INTEGRITY attribute MUST be inserted into the message as the last attribute (with dummy content) prior to the computation of the integrity check. Once the computation is performed, the value of the attribute can be filled in. This ensures the length has the correct value when the hash is performed.

10.3.4. ERROR-CODE

TOC

The ERROR-CODE attribute is present in error responses. It is a numeric value in the range of 100 to 699 plus a textual reason phrase encoded in UTF-8, and is consistent in its code assignments and semantics with [RFC3261] (Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol,” June 2002.) and [RFC2616] (Fielding, R., Gettys,

J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.). The reason phrase is meant for user consumption (typically freeform fields in alarms and logs), and can be anything appropriate for the response code. Recommended reason phrases for the defined response codes are presented below.
To facilitate processing, the class of the error code (the hundreds digit) is encoded separately from the rest of the code.

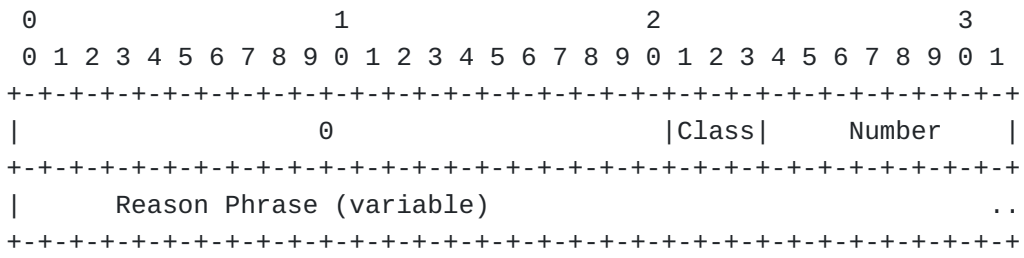

Figure 12: ERROR-CODE Syntax

The class represents the hundreds digit of the response code. The value MUST be between 1 and 6. The number represents the response code modulo 100, and its value MUST be between 0 and 99.
If the reason phrase has a length that is not a multiple of four bytes, it is padded for encoding into the message, in which case the attribute length represents the length of the entire ERROR-CODE attribute (including the reason phrase) prior to padding.
The following response codes, along with their recommended reason phrases (in brackets) are defined at this time:

400 (Bad Request): The request was malformed. The requestor should not retry the request without modification from the previous attempt.

431 (Integrity Check Failure): The request contained a MESSAGE-INTEGRITY attribute, but the HMAC failed verification. This could

be a sign of a potential attack, or misconfiguration of the password .

436 (Unknown Username): The username was not known. This was likely due to a misconfiguration.

471 (Bad Client Handle): The client handle provided in the Register request is not known.

472 (Version Number Too Low): The client published a service whose version was lower than the currently held one by the server.

474 (Unregistered): The client tried an operation, such as publish or subscribe, but it has not yet registered.

476 (Unknown Subscription): The referenced subscription does not exist.

477 (Already Registered): The client tried an initial Register request, but it is already registered.

478 (Unsupported Protocol Version): The server does not support the protocol version requested by the client.

481 (Publication Failed): The publication was attempted but could not be performed due to an error reaching the DHT. The client should try again.

10.3.5. Client-Name

TOC

The Client-Name attribute is included the Register request. It contains a textual description, in UTF-8, of the software being used by the client, including manufacturer and version number. The attribute has no impact on operation of the protocol, and serves only as a tool for diagnostic and debugging purposes. The value of Client-Name is variable length. If the value of Client-Name is not a multiple of four bytes, it is padded for encoding into the VAP message, in which case the attribute length represents the length of the attribute prior to padding. However, it MUST be less than 255 characters and MUST be at least one character long.
It is RECOMMENDED that it be constructed as:

<vendor>/<product>/<version>/<hostname or IP>

Where version includes major, minor and build.

10.3.6. Client-Handle

TOC

This attribute has a 32 bit value, representing an unsigned integer to be used as the client handle.

10.3.7. Protocol-Version

TOC

This attribute is 32 bits, consisting of two 16-bit unsigned integers, representing the major and minor version numbers of the protocol:

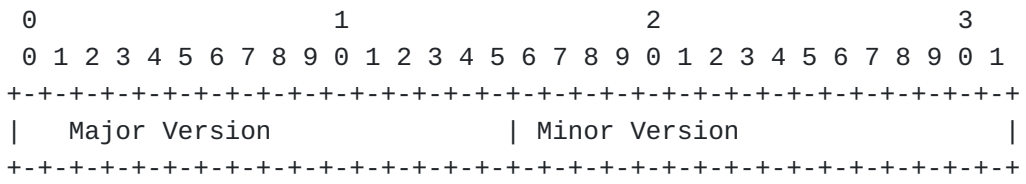

Figure 13: Protocol-Version Syntax

10.3.8. Client-Label

TOC

This attribute is a UTF-8 string, which MUST be between 1 and 255 characters. It is not used by this specification.

10.3.9. Keepalive

TOC

This attribute is a 32 bit unsigned integer, representing the number of milliseconds that the server will retain client state after the last message from the client has been received.

TOC

10.3.10. ServiceIdentity

The format of the ServiceIdentity attribute is:

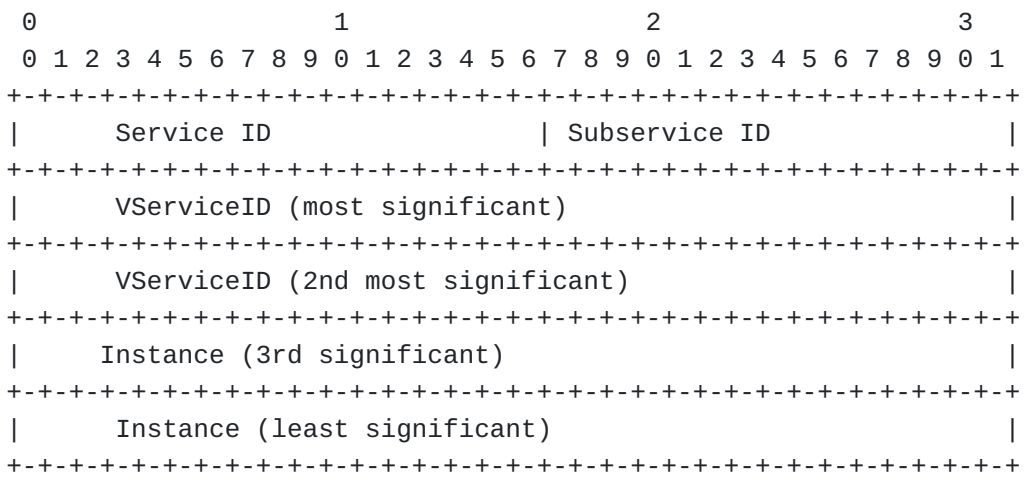

Figure 14: ServiceIdentity Attribute

The value of serviceID must always be 101. A Subservice value of 4 indicates VService publications. A subservice value of 3 indicates number publications.

10.3.11. ServiceVersion

TOC

The ServiceVersion field is a 32 bit unsigned integer. It contains the version number for the service advertised in the Publish request. It always increments by at least one for each change in the service.

10.3.12. ServiceContent

TOC

The ServiceContent is the actual content of the service definition. It is an arbitrary number of bytes. If the number of bytes of content are not a multiple of four, the content is padded with arbitrary data so that it is a multiple of four. The value of the length field of the attribute is the length prior to padding.
The ServiceContent MUST be less than 32k, despite the fact that the length field of the attribute itself would allow content up to 64k.

10.3.13. SubscriptionID

TOC

The SubscriptionID is present in successful responses to Subscribe and in Unsubscribe messages. It contains an identifier for the subscription. It is a unique handle, unique within all subscriptions between the client and this server. It is an unsigned 32 bit integer. It is also present in Notify and Withdraw requests.

10.3.14. CallDirection

TOC

This attribute is a 32 bit unsigned integer. A value of 0 indicates a received call. A value of 1 indicates a sent call. Other values are reserved and not valid in this version of the protocol.

10.3.15. StartTime

TOC

The start and is a 64 bit NTP time value. The start time is measured in the following way:

1. For calls sent to the PSTN (i.e., originated by this call agent), the start time is measured from the instant of the receipt of the call acceptance message indicating that the called party answered the call. For SIP, this would correspond to receipt of the 200 OK to the original SIP INVITE.

2. For calls received from the PSTN, (i.e., received by this call agent), the start time is measured from the instant of transmission of the call acceptance message towards the PSTN, indicating that the called party answered the call. For SIP, this would correspond to transmission of the 200 OK to the original SIP INVITE.

10.3.16. StopTime Attribute

TOC

The stop time is a 64 bit NTP value and is measured in the following way:

1. For the call agent which terminates the call, it corresponds to the transmission of the call termination message towards the

PSTN. For SIP, this corresponds to the transmission of a SIP BYE request.

2. For the call agent which receives the termination, it corresponds to the receipt of the call termination message from the PSTN. For SIP this corresponds to the receipt of a SIP BYE request.

10.3.17. CallingNum Attribute

TOC

The calling party number MUST be expressed in fully qualified E.164 format, and the attribute is a string with variable length.
The calling party number is complicated. This is because this value is often munged and modified by the PSTN as it traverses the network. Fortunately, ViPR does not depend on it being delivered or being correct, but when it is delivered it improves security. Its presence is also needed for validating numbers which connect to multiple users, such that multiple calls to that number are often in progress at the same time. For example, 800 numbers.
For the originating call agent, the value is the E.164 number of calling party number delivered to the PSTN. For the terminating call agent, the value is E.164 normalized value of the caller ID received from the PSTN. This will require that national numbers delivered over a PRI are normalized to include their country code.

10.3.18. CalledNum Attribute

TOC

The called party number MUST be expressed in fully qualified E.164 format, and it is represented in the attribute as a string with variable length. The following rules apply for computation of the called party number:
For the call agent which initiates the call, the called party number is the E.164 number, including the leading plus, of the target of the call. Of course, this may not (and is probably not) the same as the digit sequence dialed by the calling party. The originating call agent MUST normalize this number to E.164 format based on its local dialing rules.
For the call agent which receives the call, the called party number is the E.164 number, including the leading plus, of the target of the call. Of course, this may not (and is probably not), the same as the called party number as delivered by the PSTN. It is likely that country codes, for example, are omitted from the message delivered by the PSTN.

It is the responsibility of the terminating call agent to reconstruct the E.164 number of the called party.

10.3.19. Quota Attribute

TOC

This attribute consists of two 32 bit values. The first is the quota limit, which is the total number of numbers that can be published by this and other call agents attached to this ViPR server into this DHT. The second is the current total number of numbers being published by this and other call agents attached to this ViPR server into this DHT. If the current value is less than the quota value, everything is fine. Once it exceeds it, the DHT is likely to begin dropping entries and the admin needs to reduce the number of numbers being published.

10.3.20. DHTLifetime Attribute

TOC

This attribute is a 32 bit unsigned integer. It indicates the number of seconds that data written into the DHT will remain in the DHT prior to being expunged.

---

11. Security Considerations

TOC

11.1. Outsider Attacks

TOC

VAP prevents against traditional outsider attacks by means of TLS along with password-based digest authentication. That mechanism MUST be implemented by clients and servers and SHOULD be used.

11.2. Insider Attacks

TOC

Of much more concern are attacks whereby the client is authenticated, but it misuses the VAP connection to attack the overall system. The principal attack to be considered is where an attacker injects false numbers by sending Publish requests for the number service containing numbers that the client doesn't own. This attack is the

fundamental security problem that ViPR overall addresses with the validation mechanism, and so that attack is handled outside of VAP. Another potential attack is a flooding attack where a client sends a large amount of numbers into the DHT. This attack is prevented by the distributed quota mechanism within the ViPR RELOAD usage, and thus prevented outside of VAP. Similarly, an attacker might try to DOS the ViPR network by sending a large volume of reads or writes into the DHT. This is prevented by means of the rate control mechanisms enforced by the ViPR server.

---

12. IANA Considerations

TOC

There are no IANA considerations associated with this specification.

---

13. References

TOC

13.1. Normative References

TOC

[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC3261]	Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol,” RFC 3261, June 2002 (TXT).
[RFC5389]	Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for NAT (STUN),” RFC 5389, October 2008 (TXT).
[RFC0791]	Postel, J., “Internet Protocol,” STD 5, RFC 791, September 1981 (TXT).
[RFC2617]	Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” RFC 2617, June 1999 (TXT, HTML, XML).
[RFC2104]	Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” RFC 2104, February 1997 (TXT).
[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” RFC 2616, June 1999 (TXT, PS, PDF, HTML, XML).

[VIPR-PVP]	Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "The Public Switched Telephone Network (PSTN) Validation Protocol (PVP)," draft-rosenberg-dispatch-vipr-pvp-03 (work in progress), October 2010 (TXT).

13.2. Informative References

TOC

[VIPR-OVERVIEW]	Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Verification Involving PSTN Reachability: Requirements and Architecture Overview," draft-rosenberg-dispatch-vipr-overview-04 (work in progress), October 2010 (TXT).

Appendix A. Release notes

TOC

This section must be removed before publication as an RFC.

A.1. Modifications between rosenberg-03 and rosenberg-02

TOC

*Nits.

*Shorter I-Ds references.

*Added terminology section.

*Changed figures to fit in the page width.

*Change reference from RFC 2401 to RFC 2104

*Removed cut & paste error from STUN.

*Fixed some invalid lists.

*Section 9.1: Removed mutual authentication to be consistent with 5.1.

*Fixed the text for the creation of the resource name in 9.4.2, to be consistent with -reload-usage.

*Fixed example to really contain hexadecimal.

Authors' Addresses

TOC

	Jonathan Rosenberg
	jdrosen.net
	Monmouth, NJ
	US
Email:	jdrosen@jdrosen.net
URI:	http://www.jdrosen.net
	Cullen Jennings
	Cisco
	170 West Tasman Drive
	MS: SJC-21/2
	San Jose, CA 95134
	USA
Phone:	+1 408 421-9990
Email:	fluffy@cisco.com
	Marc Petit-Huguenin
	Stonyfish
Email:	marc@stonyfish.com