International

Virtual

Observatory

Alliance

VOEvent Transport Protocol

Version 1.2 (DRAFT #4)

IVOA Note 2013 May 13

Previous version(s):
Version 1.1 (2009 August 5)
Version 1.0 (2009 May 14)

Author(s):
Alasdair **Allan**, Babilim Light Industries, Exeter, UK
Robert B. **Denny**, DC-3 Dreams, SP, Mesa, AZ, USA
John **Swinbank**, University of Amsterdam, The Netherlands

Abstract

The IVOA VOEvent Recommendation (Seaman, et al., 2011) defines a means of describing a transient celestial event but, purposely, remains silent on the topic of how those descriptions should be transmitted. This document formalizes a TCP-based protocol for VOEvent transportation that has been in use by members of the VOEvent community for several years and discusses the topology of the event distribution network. It is intended to act as a reference for the production of compliant protocol implementations.

Status of This Document

This document has been produced by John Swinbank based on the version 1.1 specification by Allan & Denny. This draft version is a work in progress. It is intended to add clarifications to the previous version without changing the on-the-wire protocol.

This is an IVOA Note expressing suggestions from and opinions of the authors. It is intended to share best practices, possible approaches, or other perspectives on interoperability with the Virtual Observatory. It should not be referenced or otherwise interpreted as a standard specification.

A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/documents/.

Acknowledgements

- Robert W. White, *ex* LANL, now at NREL
- Phillip Warner, *ex* NOAO
- Robert Seaman, NOAO

1 Introduction

The VOEvent standard (Seaman, et al., 2011) defines a means of representing a transient celestial event with an implicit request for action on the part of the recipient. The VOEvent standard is transport neutral: it does not take a position on the mechanism by which the event should be transmitted from its author to interested recipients. However, it encourages the construction of "a robust general-purpose network of interoperating brokers" for event transmission.

To date, a number of different event distribution networks have been prototyped and met with varying degrees of technical success and community adoption. However, as the number of interested participants grows, and next-generation large-scale survey instruments such as LSST[1], LIGO[2], LOFAR[3] and SKA[4] are developed and begin to become available, it is clear that a standard, interoperable mechanism for event communication is required. It is such a mechanism that this document aims to describe.

The purpose of the protocol described herein is to transport a VOEvent message from its sender to one or more interested recipients. To achieve this, we envision three distinct network roles: *authors*, which create events; *brokers*, which receive events from authors and distribute them, and *subscribers*, which receive and (if appropriate) act upon the events. See Figure 1 for an illustration. Note that a single entity may perform more than one role within the network: for example, creating events and distributing its own creations (combining the author and broker roles) or receiving events from a broker and redistributing them to a list of subscribers (combining the subscriber and broker roles).

Building upon this architecture, a strongly-connected set of brokers which subscriber to each other's event streams and redistribute to their subscribers (the "VOEventNet backbone") provides a fault-tolerant system which is resilient against the failure of one or more network entities. Such a backbone system is already under construction by members of the VOEvent community.

The protocol described herein is intentionally as simple as possible while still accomplishing the required task. There are some value-added VOEvent services being developed which may require more complex protocols, but these fall outside the scope of the current document.

[1] Large Synoptic Survey Telescope; http://www.lsst.org/
[2] Laser Interferometric Gravitational-Wave Observatory; http://www.ligo.org/
[3] Low Frequency Array; http://www.lofar.org/
[4] Square Kilometre Array; http://www.skatelescope.org/

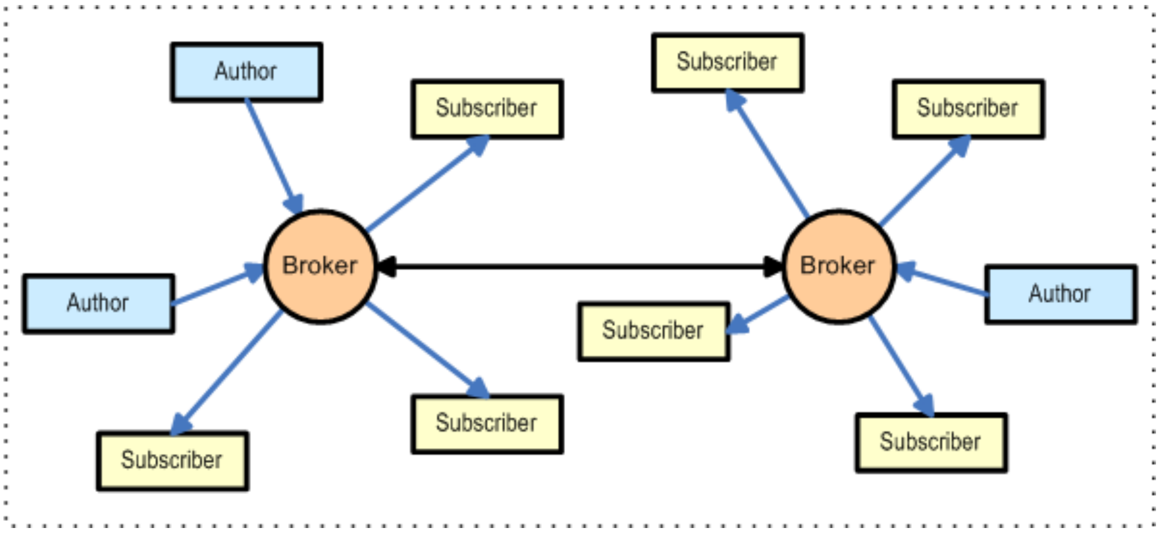

Figure 1. VOEvent network architecture showing node roles

2 Terminology

Throughout this document, we adopt the terminology of RFC 2119 (Bradner, 1997). In particular, note that:

- The word "must" indicates an absolute requirement of the specification;
- The word "should" indicates behavior that is normally included in implementations of the specification, but there may exist valid reasons for excluding it in particular circumstances;
- The word "may" indicates purely optional behavior which is permitted according to this specification.

3 Common Characteristics

3.1 Design Goals

The Transport Protocol, hereafter VTP, is provides a simple means of transporting VOEvents from authors through brokers to subscribers.

VTP transmits a up to single VOEvent message in each transaction. If multiple VOEvent messages are to be transmitted, multiple transactions must take place.

VTP is non-transformational on VOEvents being transmitted: the message delivered to a subscriber should be bit-for-bit identical to that provided by an author. If an intermediary wishes to modify or annotate the event, they should not edit the event in transport, but rather generate a new event to supplement or replace it.

VTP does not provide a transport-level means of annotating or otherwise embellishing VOEvents, or of providing stream-level metadata.

VTP values simplicity of design and operation to lower the barrier to entry. It is not intended to meet every use case. As per Section 1, it is anticipated that some VOEvent-based services will require more complex protocols.

3.2 Network Layer

VTP operates over TCP (Information Sciences Institute, 1981) connections, and relies on TCP's guaranteed error-free in-order delivery of data: no checksum or digest data is included. All messages are sent over the TCP connection preceded by a 4-byte network-ordered [5] count, followed immediately by the payload data. The 4-byte count is interpreted as a 32-bit integer equal to the number of payload bytes *following* the count bytes. The payload is considered an opaque collection of bytes at this level[6].

3.3 Message Format

The payload of the a VTP message is an XML document. It must consist of an XML declaration followed by, in order, optional XML comments, a single VOEvent or Transport element, and more optional XML comments. It must validate against either the VOEvent XML schema[7] or the Transport XML schema (page 17). Transport elements are used by the VTP system itself and are invisible to end-users: see Section 6 for details.

3.4 Broker Behavior

Although the simplest broker implementation may simply forward all unique events it receives, either directly from authors or from other brokers, to all of its subscribers, this behavior is not required. Instead, the broker may provide “value-added” services which limit how messages are redistributed. For example, a broker may make arrangements with some or all of its subscribers to filter the events it receives, and only forward those events fulfilling some predefined criteria to its subscribers. Similarly, brokers may limit access to some clients based on various criteria (see Section 9 for details).

VTP does not provide in-band notification of these per-broker details. For example, the protocol does not make an author submitting to a filtering broker aware that their event might not be sent to all of the broker’s subscribers, and, similarly, it does not make a subscriber of a filtering broker aware that they might not receive a complete set of events. It is the responsibility of authors and subscribers to ensure that the brokers they use provide the services they require. Brokers should clearly advertise any added-value behavior they provide, for example on a website or through the IVOA registry.

[5] As defined by RFC 1700 (Reynolds & Postel, 1994), also called “big endian” ordering. The Berkeley sockets API defines functions *htonl()* and *ntohl()* which convert (as needed) between the local processor-native byte ordering and the network-ordered standard.
[6] As a result, the format of the document being transmitted is opaque to the transport layer. Therefore both ASCII and UTF-8 are equally supported.
[7] http://www.ivoa.net/xml/VOEvent/VOEvent-v2.0.xsd

4 Node Roles

The VOEvent network consists of three types of nodes (refer to Figure 1, above):

- Author
- Broker
- Subscriber

The flow of messages is over two types of connections:

- Author to Broker
- Broker to Subscriber

Each type of connection is discussed qualitatively below.

4.1 Author to Broker

When an author wants to submit a VOEvent message to the network, it opens a TCP connection to a broker, sends the message, waits for a response from the broker, and then closes the TCP connection. The response from the broker is a Transport message.

4.2 Broker to Subscriber

When a subscriber wants to receive VOEvent message traffic, it opens a TCP connection to a broker. This connection is kept open continuously. When the broker receives a message, it relays a copy of the VOEvent message to all of the connected subscribers. Thus, a subscriber must continuously listen on the TCP connection and be prepared to receive VOEvent messages at any time, even when it is busy processing a previously received VOEvent message. When a subscriber receives a VOEvent message from its broker, it must respond with an appropriate Transport message.

4.3 Broker to Broker

Traffic between brokers uses the preceding methods. Each broker takes the role subscriber as far as every other broker is concerned. A broker that wishes to receive a feed from another broker should connect to that broker's subscriber port. No special protocol features are needed.

5 Connection Maintenance

All connections over which a *broker* sends VOEvent messages are kept open continuously. Basic TCP does not provide any dead-peer indication[8]. Further, network infrastructure devices might sever a TCP connection after some period of inactivity. This gives rise to the need for keep-alive messages. After

[8] TCP does support a *keep-alive* service, but it is considered by some to be controversial and is not accessible from some socket APIs (Braden, 1989).

no more than 90 seconds of inactivity on any given connection, the broker must send a Transport `iamalive` message, to which the subscriber must reply with a copy of that message plus some optional identification information. For details of the message format, see Section 6.

At both ends of the continuous connection, the node either expects to receive an `iamalive` message or expects to receive the response to its `iamalive` message. If not seen, the node should assume that the connection has been lost or the peer is dead. At this point, the node that was responsible for opening the connection may attempt to re-initiate it. A geometric back-off algorithm should be used to avoid network load.

6 Transport Messages

Transport messages are XML documents. There are four classes of Transport message:

- `iamalive` (Connection maintenance)
- `authenticate` (Authentication request/response)
- `ack` (VOEvent successful receipt acknowledgement)
- `nak` (VOEvent unsuccessful receipt acknowledgement)

All Transport messages have the same general syntax, and are defined by the Transport schema (see Appendix A: Transport Schema). The `role` attribute of the root `<Transport>` element distinguishes between the types listed above. The connection maintenance and receipt acknowledgement message types are described in detail in this section; the authentication message type has a special role which is described in Section 9.2.2.

6.1 `iamalive` Message

The *iamalive* message is indicated by a role equal to “iamalive”. The `<Origin>` element contains the IVORN[9] of the broker which is managing the connection. The `<TimeStamp>` element must contain the UTC date/time at which the message was generated.

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

<trn:Transport role="iamalive" version="1.0"
 xmlns:trn="http://telescope-networks.org/schema/Transport/v1.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://telescope-networks.org/schema/Transport/v1.1
                    http://telescope-networks.org/schema/Transport-v1.1.xsd">
    <Origin>ivo://uk.org.estar/estar.ex#</Origin>
    <TimeStamp>2009-04-09T22:39:06</TimeStamp>
</trn:Transport>
```

Figure 2. Sample `iamalive` message

[9] International Virtual Observatory Resource Name (Plante, Linde, Williams, & Noddle, 2007).

6.2 `iamalive` Response

The `iamalive` response is an extension of the initial `iamalive` message. It also has a role of `iamalive`. It must include an additional `<Response>` element containing the IVORN of the subscriber. It may include a `<Meta>` element with `<Param>` sub-elements which give *additional* information about the subscriber or any other relevant information. `<Param>` elements have no content and must contain `name` and `value` attributes. The names and values may be any string. The `<TimeStamp>` element contains the date/time (UTC) at which the response is sent.

```
<?xml version='1.0' encoding='UTF-8'?>

<trn:Transport role="iamalive" version="1.0"
 xmlns:trn="http://telescope-networks.org/schema/Transport/v1.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://telescope-networks.org/schema/Transport/v1.1
                     http://telescope-networks.org/schema/Transport-v1.1.xsd">
    <Origin>ivo://uk.org.estar/estar.ex#</Origin>
    <Response>ivo://com.dc3/engineering#</Response>
    <TimeStamp>2009-04-09T22:39:07</TimeStamp>
    <Meta>
        <Param name="IPAddr" value="123.123.123.123" />
        <Param name="Contact" value="rdenny@dc3.com" />
    </Meta>
</trn:Transport>
```

Figure 3. Sample `iamalive` response message

6.3 VOEvent Message Receipt Response

The VOEvent message receipt response is similar to the `iamalive` response except the role is either `ack` or `nak`, the `<Origin>` is the IVORN of the just-received VOEvent message, and an optional `<Result>` element may accompany the `<Param>` elements. `<Result>` may contain any string, and it is recommended that it contain a human-readable error message if role is `nak`. The `<TimeStamp>` element contains the date/time (UTC) at which the response is sent.

The `nak` response indicates that the recipient is unable or unwilling to take responsibility for this message. This may be because, for example, the message fails to validate as a valid VOEvent, or because it was received from an unauthorized client (see Section 9). A `nak` response is not appropriate if the sender is able to accept the message but then decides not to redistribute it (for example, if it is a duplicate of an event which has already been distributed: see Section 8).

```
<?xml version='1.0' encoding='UTF-8'?>

<trn:Transport role="ack" version="1.0"
 xmlns:trn="http://telescope-networks.org/schema/Transport/v1.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://telescope-networks.org/schema/Transport/v1.1
                     http://telescope-networks.org/schema/Transport-v1.1.xsd">
    <Origin>ivo://nvo.caltech/voeventnet/catot#901211380084129246</Origin>
    <Response>ivo://com.dc3/engineering#</Response>
    <TimeStamp>2009-04-09T22:54:01</TimeStamp>
    <Meta>
        <Param name="IPAddr" value="123.123.123.123" />
        <Param name="Contact" value="rdenny@dc3.com" />
        <Result>Message received and validated successfully</Result>
    </Meta>
</trn:Transport>
```

Figure 4. Sample VOEvent message receipt response (`ack`)

```
<?xml version='1.0' encoding='UTF-8'?>

<trn:Transport role="nak" version="1.0"
 xmlns:trn="http://telescope-networks.org/schema/Transport/v1.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://telescope-networks.org/schema/Transport/v1.1
                     http://telescope-networks.org/schema/Transport-v1.1.xsd">
    <Origin>ivo://nvo.caltech/voeventnet/catot#901211380084129246</Origin>
    <Response>ivo://com.dc3/engineering#</Response>
    <TimeStamp>2009-04-09T22:54:01</TimeStamp>
    <Meta>
        <Param name="IPAddr" value="123.123.123.123" />
        <Param name="Contact" value="rdenny@dc3.com" />
        <Result>Error in VOEvent message: ISOTime not in ISO 8601 format</Result>
    </Meta>
</trn:Transport>
```

Figure 5. Sample VOEvent message receipt response (`nak`)

7 Protocol Operation

This section describes the operation and sequencing of VTP operation for each end of a connection between an author and a broker, as well as between a broker and a subscriber. See Section 4 above for a qualitative discussion of the protocol from the viewpoint of each entity. In the sections below, the “message” is the combination of the byte count and payload as described in Section 3.3 above.

The protocol diagrams in the following sections depict *layers* between which are *interfaces*. Figure 6 shows the overall architecture of communication between authors, brokers, and subscribers. Each entity uses the transport layer to send and/or receive VOEvent messages.

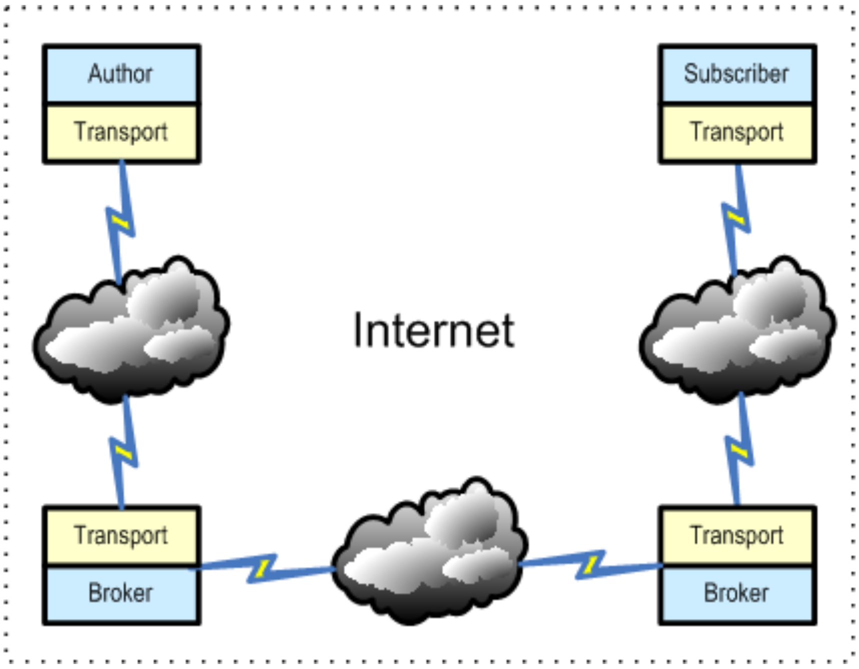

Figure 6. Layered architecture

7.1 Author Sending to Broker

The author provides a single, complete event to the transport layer for transmission. It is the responsibility of the author to apply a digital signature (Section 9.2) to the message if required before sending it to the transport layer. It is the responsibility of the transport layer to prepend the byte count (Section 3.3) to the message before sending it to the broker, and to interpret the `ack/nak` message (Section 6.3) from the broker. The transport layer returns a value indicating success or failure to the author; it is the responsibility of the author to check this and to initiate any appropriate responsive action. See Figure 7 for details.

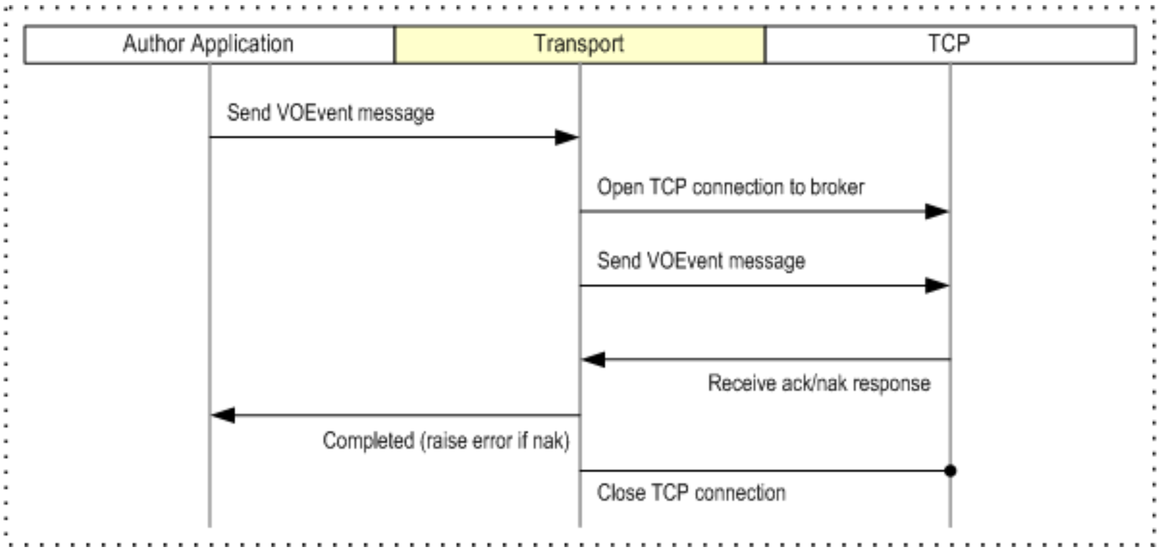

Figure 7. Transport protocol at an author node

7.2 Broker Receiving from Author

The broker responds to two events from the transport layer, as shown in Figure 8. One event indicates that a request for a TCP connection has arrived, and the broker must respond with accept or reject status. This is where the broker can provide IP whitelist access control (Section 9.1). The other event indicates that a message has been received, and the broker must respond with accept or reject status. This is where the broker can validate the XML and check any digital signature (Section 9.2.1). It is the responsibility of the Transport layer to remove the byte count (Section 3.3) from the incoming message, and to create the `ack/nak` response (Section 6.3).

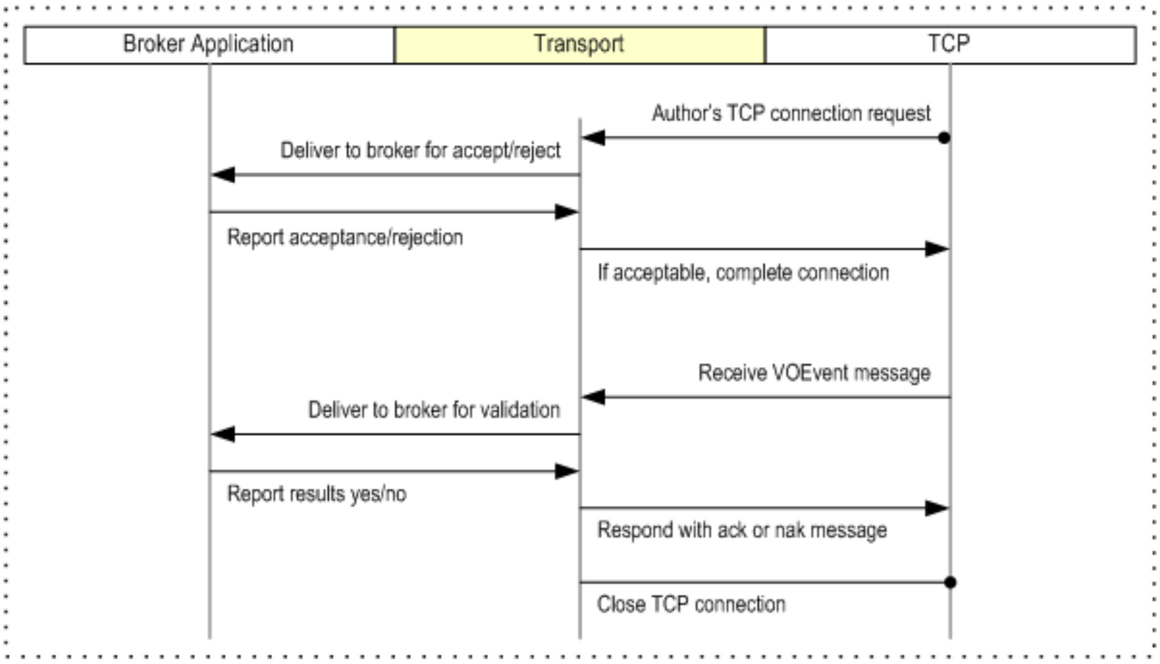
Broker Application
Transport
TCP
Author's TCP connection request
Deliver to broker for accept/reject
Report acceptance/rejection
If acceptable, complete connection
Receive VOEvent message
Deliver to broker for validation
Report results yes/no
Respond with ack or nak message
Close TCP connection

Figure 8. Transport protocol at broker, receiving from author

7.3 Broker Sending to Subscriber

See Figure 9. Subscribers connect to the broker and leave the connection open indefinitely. The broker responds to an event from the transport layer indicating that a subscriber wants to open a TCP connection, and the broker must respond with accept or reject status. This is where the broker can provide IP white list access control (Section 9.1). If accepted, the broker would presumably add this subscriber to its distribution list.

Once the subscriber has successfully connected to the broker, the broker's Transport layer begins sending periodic `iamalive` messages to the subscriber, who replies with an `iamalive` response (Sections 6.1 and 6.2). If the subscriber's `iamalive` reply messages stop arriving, the Transport layer may assume that the subscriber is dead or gone, and close the TCP connection. The `iamalive` process continues until the subscriber closes the TCP connection or the broker's Transport layer stops receiving `iamalive` replies from the subscriber.

To send a message to the subscriber, the broker provides it to the transport layer. It is the responsibility of the transport layer to prepend the byte count (Section 3.3) to the message before sending to the subscriber, and to interpret the resulting ack/nak message (Section 6.3). The transport layer returns a value indicating success or failure to the broker; in the event of an error (`nak` received, dead-peer, or subscriber closed the connection), the broker may remove the subscriber from its distribution list.

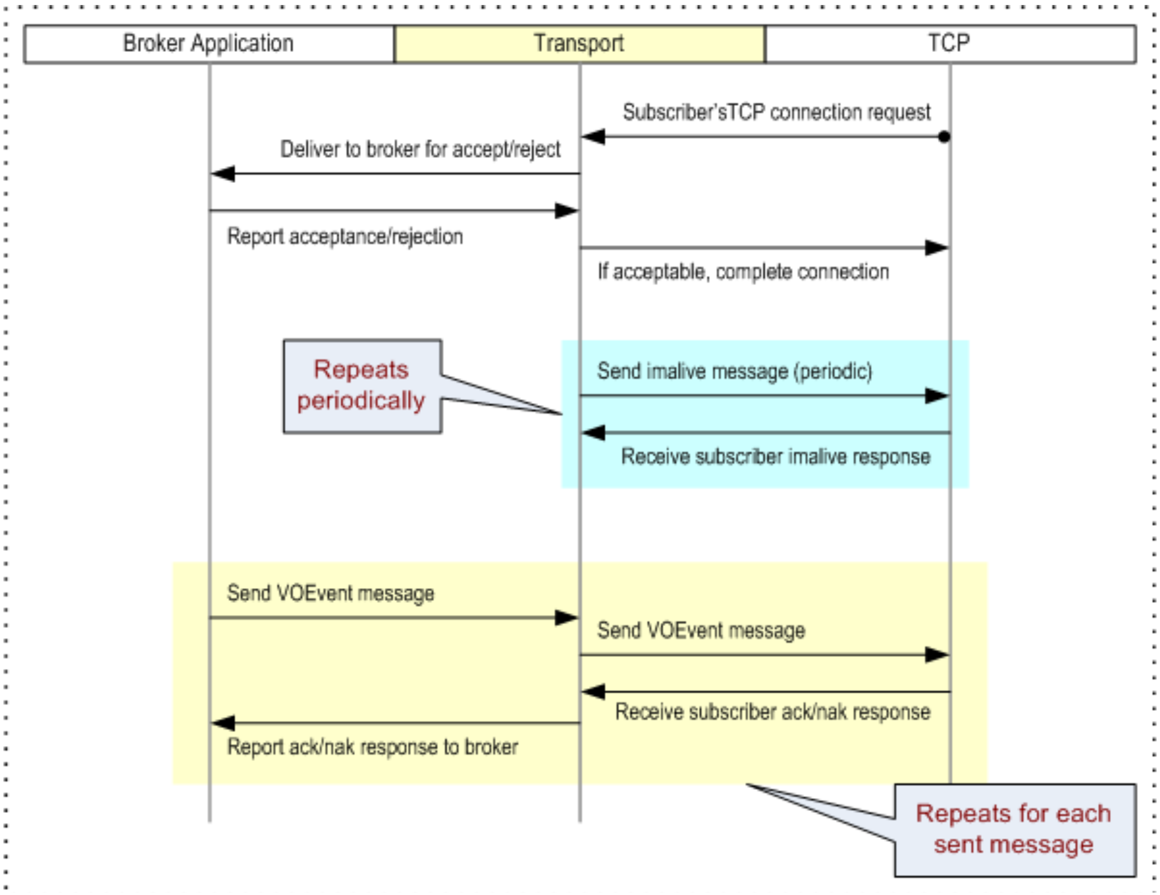
Broker Application
Transport
TCP
Subscriber'sTCP connection request
Deliver to broker for accept/reject
Report acceptance/rejection
If acceptable, complete connection
Repeats periodically
Send imalive message (periodic)
Receive subscriber imalive response
Send VOEvent message
Send VOEvent message
Receive subscriber ack/nak response
Report ack/nak response to broker
Repeats for each sent message

Figure 9. Transport protocol at broker, sending to subscriber

7.4 Subscriber Receiving from Broker

See Figure 10. Subscribers connect to the broker and leave the connection open indefinitely. If the connection is successful, the subscriber will begin receiving messages.

As per Section 5, when the subscriber has successfully connected to the broker, the subscriber's Transport layer begins listening for periodic `iamalive` (Section 6.1) messages from the broker. For each `iamalive` message received, the Transport layer responds with an `iamalive` *reply* (Section 6.2). If the broker's `iamalive` messages stop arriving, the subscriber's Transport layer may assume that the connection has been lost and attempt to re-connect to the broker. Reconnection attempts should use a geometric back-off algorithm. If the subscriber's Transport layer cannot re-establish the connection, it should notify the subscriber of the error.

The subscriber client responds to an event advising that a (VOEvent) message has arrived, and the subscriber client must respond with accept or reject status. This is where the subscriber can validate the XML and check any digital signature that may be present. It is the responsibility of the Transport layer to remove the byte count (Section 3.3) from the incoming message, and to create and send the `ack/nak` message (Section 6.3).

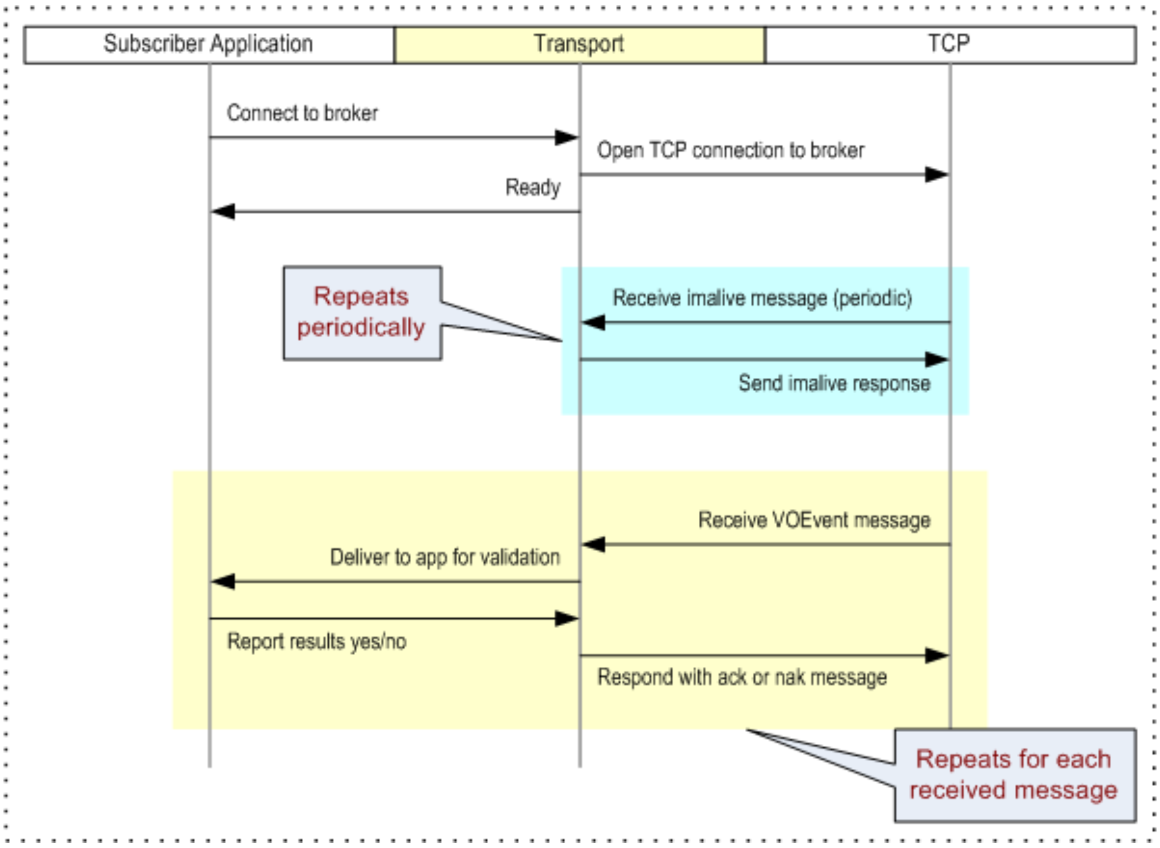
Subscriber Application
Transport
TCP
Connect to broker
Open TCP connection to broker
Ready
Repeats periodically
Receive imalive message (periodic)
Send imalive response
Receive VOEvent message
Deliver to app for validation
Report results yes/no
Respond with ack or nak message
Repeats for each received message

Figure 10. Transport protocol at subscriber

8 De-duplication

In a network topology like that illustrated in **Error! Reference source not found.**, multiple brokers service independent sets of authors and subscribers. As per Sections 1 and 4.3, brokers will subscribe to each other's event feeds to ensure that their subscribers have access to the full range of available events. In this way, in a complex VOEvent network, many brokers will form one or more strongly connected sets.

In this situation, there is a risk of event loops developing on the network: broker A receives an event from B and forwards it to its subscriber list, which includes A, which forwards it to its subscriber lists, which includes B, and so on.

In order to prevent this, each broker must process each unique event it receives a maximum of once. For this purpose, we regard “unique” as meaning that the content between the opening < and closing > of the `<VOEvent>` element is bit-for-bit identical, including all white space characters[10]. Note that it is now established practice to distribute different serializations (eg VOEvent 1.1 and 2.0) of the same event with the same IVORNs[11]. Consequently, an IVORN is not a unique identifier of a particular event representation, and is not, therefore, suitable for use in network de-duplication.

9 Limiting Access

For administrative or security reasons, broker administrators may wish to limit access to the services they provide to a restricted range of clients. These restrictions may be required on either or both of the connection types in VTP: author to broker (i.e., a limit on which authors can publish through a given broker) or broker to subscriber (a limit on which subscribers a broker is willing to provide with event streams).

Two mechanisms are available within the VTP system to address these requirements.

9.1 IP Address White-Listing

If the broker knows a priori the IP addresses or ranges from which authorized authors or subscribers are permitted to connect (a “whitelist”), it may simply deny connections from addresses which fall outside that range. If the sets of authorized authors and subscribers are not the same, separate whitelists may be implemented.

This mechanism is simple and effective. However, it imposes significant administrative overhead on the broker owner if large and complex whitelists are required. Further, it is of limited applicability if the clients to be serviced are using dynamic IP addresses (that is, addresses which change periodically).

9.2 Cryptographic Signatures

A cryptographic signature can be used to verify the identity of the sender of a message. A simple and transparent means to digitally sign XML messages, with particular reference to VOEvent, has been developed to be compatible with the VTP system (Denny, 2008). Brokers can use this scheme to check the identity of clients. The mechanism varies slightly depending on whether the client is an author or a subscriber.

[10] The implementation of this check is at the discretion of the broker. Appropriate techniques may include directly comparing the bitstream (which would necessarily mean storing an archive of previously-processed events) or calculating a hash function such as SHA1 (Jones & Eastlake 3rd, 2001) over the event contents and storing the result.
[11] http://www.ivoa.net/pipermail/voevent/2012-March/002836.html and subsequent discussion.

9.2.1 Author Submission

Authors connect, send a VOEvent message, and then disconnect (see Section 7.1). All that's needed is for the author to sign the VOEvent message being submitted. The broker's acceptance/rejection criteria include validating the signature. If that succeeds, the broker accepts the message and returns an `ack`. Otherwise, if the broker does not have the author's public key or if validation fails, the message is discarded and the broker returns a `nak`. A `<Result>` element may be included explaining that the signature validation failed: see Section 6.3.

The benefits of having authors sign their VOEvent messages extend beyond broker access control: the presence of a signature on an author's VOEvent message can be used by subscribers to guarantee the integrity of the message. Note, however, that the subscriber must not assume that brokers have checked all signatures on their behalf: if the subscriber requires that the event integrity be verified, they must check the signature themselves.

9.2.2 Subscriber Authentication

Subscribers do not send VOEvent messages to the broker (see Section 7.4). To accommodate subscriber authentication using a signed message, the broker sends a Transport `authenticate` message (see the example in Figure 11) to the subscriber immediately after the subscriber connects. The subscriber returns a digitally signed `authenticate` response, with which the broker authenticates the subscriber as before. If the authentication succeeds, the broker leaves the TCP connection open and begins to deliver VOEvent messages to the subscriber. If the authentication fails, the broker simply closes the TCP connection. There is no need for the subscriber to digitally sign subsequent `iamalive` responses as long as the TCP connection remains open. If the connection is lost, forcing a reconnect, the `authenticate` exchange will occur as before.

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

<trn:Transport role="authenticate" version="1.0"
 xmlns:trn="http://telescope-networks.org/schema/Transport/v1.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://telescope-networks.org/schema/Transport/v1.1
                     http://telescope-networks.org/schema/Transport-v1.1.xsd">
    <Origin>ivo://uk.org.estar/estar.ex#</Origin>
    <TimeStamp>2009-04-09T22:39:06</TimeStamp>
</trn:Transport>
```

Figure 11. Sample authenticate message

Appendix A: Transport Schema

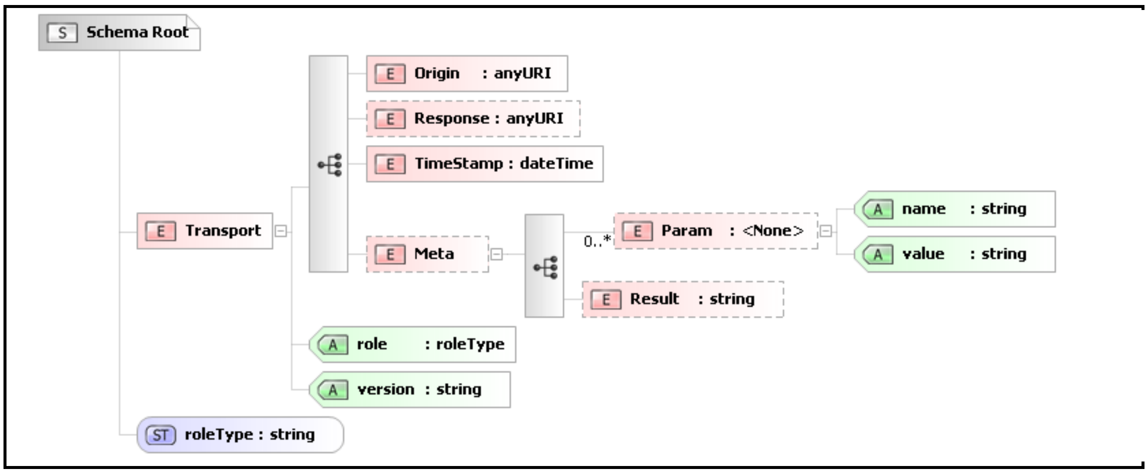

Figure 12. Graphical representation of transport schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="Transport">
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="1" name="Origin" type="xs:anyURI" />
    <xs:element minOccurs="0" maxOccurs="1" name="Response" type="xs:anyURI" />
    <xs:element minOccurs="1" name="TimeStamp" type="xs:dateTime" />
    <xs:element minOccurs="0" maxOccurs="1" name="Meta">
     <xs:complexType>
      <xs:sequence minOccurs="1" maxOccurs="1">
       <xs:element minOccurs="0" maxOccurs="unbounded" name="Param">
        <xs:complexType>
         <xs:attribute name="name" type="xs:string" use="required" />
         <xs:attribute name="value" type="xs:string" use="required" />
        </xs:complexType>
       </xs:element>
       <xs:element minOccurs="0" maxOccurs="1" name="Result" type="xs:string" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="role" type="roleType" use="required" />
   <xs:attribute name="version" type="xs:string" use="required" />
  </xs:complexType>
 </xs:element>
 <xs:simpleType name="roleType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="iamalive" />
   <xs:enumeration value="authenticate" />
   <xs:enumeration value="ack" />
   <xs:enumeration value="nak" />
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

Figure 13. Transport schema

Appendix B: Version History

Revised in V1.2

Version 1.2 adds two new sections to the document: Section 3.1, outlining design goals of the protocol, and Section 8, detailing requirements for message de-duplication to avoid network loops. In addition, it adds an explicit time requirement to connection maintenance packets (Section 5) and clarifies the semantics of `nak` transport messages (Section 6.3).

Revised in V1.1

Version 1.1 adds an optional `<Result>` sub-element (containing text) within the optional `<Meta>` element. This is intended to convey details on errors encountered if the Transport response is `nak` but may also be used for informational purposes in `ack` messages.

Bibliography

Braden, R. (1989). *Request for Comments 1122: Requirements for Internet Hosts — Communication Layers.* Internet Engineering Task Force.

Bradner, S. (1997). *Request for Comments 2119: Key words for use in RFCs to Indicate Requirement Levels.* Internet Engineering Task Force.

Denny, R. (2008). *A Proposal for Digital Signatures in VOEvent Messages.* International Virtual Observatory Alliance.

Information Sciences Institute. (1981). *Request for Comments 793: Transmission Control Protocol.* Defense Advanced Research Projects Agency.

Jones, P., & Eastlake 3rd, D. (2001). *Request for Comments 3174: US Secure Hash Algorithm 1 (SHA1).* Internet Engineering Task Force.

Plante, R., Linde, T., Williams, R., & Noddle, K. (2007). *IVOA Identifiers, Version 1.12.* International Virtual Observatory Alliance.

Reynolds, J., & Postel, J. (1994). *Request for Comments 1700: Assigned Numbers.* Internet Engineering Task Force.

Rixon, G. (2005). *Single-Sign-On Authentication for the IVO: introduction and description of principles.* International Virtual Observatory Alliance.

Seaman, R., Williams, Roy, Allan, A., Barthemly, S., Bloom, J. S., Brewer, J. M., et al. (2011). *Sky Event Reporting Metadata (VOEvent), Version 2.0.* International Virtual Observatory Alliance.