Abstract

This document describes a method that allows an email sender to specify a complaint feedback loop (FBL) address as an email header. Also it defines the rules for processing and forwarding such a complaint. The motivation for this arises out of the absence of a standardized and automated way to provide mailbox providers with an address for a complaint feedback loop. Currently, providing and maintaining such an address is a manual and time-consuming process for email senders and providers.

It is unclear, at the time of publication, whether the function provided by this document has widespread demand, and whether the mechanism offered will be adopted and found to be useful. Therefore, this is being published as an Experiment, looking for a constituency of implementers and deployers, and for feedback on the operational utility. The document is produced through the Independent RFC stream and was not subject to the IETF’s approval process.

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 https://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 5 September 2022.
Table of Contents

1.  Introduction and Motivation . . . . . . . . . . . . . . . .   3
   1.1.  Scope of this Experiment  . . . . . . . . . . . . . . . .   4
   1.2.  Difference to One-Click-Unsubscribe  . . . . . . . . . .   5
2.  Definitions . . . . . . . . . . . . . . . . . . . . . . . . .   5
3.  Requirements  . . . . . . . . . . . . . . . . . . . . . . . .   6
   3.1.  Received email  . . . . . . . . . . . . . . . . . . . . .   6
      3.1.1.  Strict  . . . . . . . . . . . . . . . . . . . . . . .   6
      3.1.2.  Relaxed  . . . . . . . . . . . . . . . . . . . . . . .  6
      3.1.3.  Complex  . . . . . . . . . . . . . . . . . . . . . . .  7
      3.1.4.  DKIM signature  . . . . . . . . . . . . . . . . . . .  8
   3.2.  Report email  . . . . . . . . . . . . . . . . . . . . . . .  8
4.  Implementation  . . . . . . . . . . . . . . . . . . . . . . .   8
   4.1.  Email senders  . . . . . . . . . . . . . . . . . . . . . .  8
   4.2.  Mailbox provider  . . . . . . . . . . . . . . . . . . . . .  9
5.  Complaint report  . . . . . . . . . . . . . . . . . . . . . . .  9
   5.1.  XARF compatible report  . . . . . . . . . . . . . . . . .  10
6.  Header Syntax  . . . . . . . . . . . . . . . . . . . . . . . .  10
   6.1.  CFBL-Address  . . . . . . . . . . . . . . . . . . . . . . .  10
   6.2.  CFBL-Feedback-ID  . . . . . . . . . . . . . . . . . . . . .  10
7.  Security Considerations  . . . . . . . . . . . . . . . . . . .  10
   7.1.  Attacks on the FBL address  . . . . . . . . . . . . . . . .  10
   7.2.  Automatic suspension of an account  . . . . . . . . . . .  11
   7.3.  Enumeration attacks / provoking unsubscribe  . . . . . . .  11
   7.4.  GDPR and other data-regulation laws  . . . . . . . . . . .  11
   7.5.  Abusing for Validity and Existence Queries  . . . . . . .  12
8.  IANA Considerations  . . . . . . . . . . . . . . . . . . . . .  12
   8.1.  CFBL-Address  . . . . . . . . . . . . . . . . . . . . . . .  12
   8.2.  CFBL-Feedback-ID  . . . . . . . . . . . . . . . . . . . . .  13
9.  Examples  . . . . . . . . . . . . . . . . . . . . . . . . . . .  13
   9.1.  Simple  . . . . . . . . . . . . . . . . . . . . . . . . . .  13
   9.2.  GDPR safe report  . . . . . . . . . . . . . . . . . . . . .  14
   9.3.  GDPR safe report with HMAC  . . . . . . . . . . . . . . .  15
1. Introduction and Motivation

For a long time there has been a way for a mailbox provider to forward manual complaints back to the email sender. The mailbox provider provides what is called a feedback loop [RFC6449]. This feedback loop is used to give operators of, e.g. broadcast marketing lists, feedback on resulting complaints from their marketing mailings. These complaints are based on manual user interaction, e.g. IMAP movement to "junk" or by clicking on a "This is spam" button.

As described in [RFC6449] the registration for such a feedback loop needs to be done manually by a human at any mailbox provider who provides a FBL. This can be quite time-consuming if there are new feedback loops rising up, or the email sender wants to add new IP addresses or DKIM domains. In addition, a manual process is not well suited and/or feasible for smaller mailbox providers.

Because of the manual process involved, the email sender has to go through all providers again, delete his existing subscriptions and register with their new complaint address.

This document addresses this issue with a new email header. It extends the recommendations for the complaint feedback loop described in [RFC6449] with an automated way to submit the necessary information to mailbox providers.

Mail senders can add this header, willing mailbox providers can use it to forward the generated report to the specified complaint address. The email sender only needs to add an email header and does not need to manually register with each feedback loop provider. The elimination of a manual registration and verification process would be another advantage for the mailbox providers.

A new email header has been chosen in favour of a new DNS record to easily distinguish between multiple broadcast marketing list operators / email senders without requiring user or administrator intervention. For example, if a company uses multiple mailing systems, each system can set this header itself without requiring any change by the users or administrators within their DNS. No additional DNS query is required on the mailbox provider side to obtain the complaint address.
This document has been prepared in compliance with the GDPR and other data protection laws to address the resulting issues when providing an automated address for a complaint feedback loop, as the email may contain personal data.

Nevertheless, the described mechanism below potentially permits a kind of man-in-the-middle attack between the domain owner and the recipient. A bad actor can generate forged reports to be "from" a domain name the bad actor is attacking and send them to the complaint FBL address. These fake messages can result in a number of actions, such as blocking of accounts or deactivating recipient addresses. This potential harm and others are described with potential countermeasures in Section 7.

In summary, this document has the following objectives:

* Allow email senders to signal that a complaint address exists without requiring manual registration with all providers.
* Allow mailbox providers to obtain a complaint address without developing their own manual registration process.
* Be able to provide a complaint address to smaller mailbox providers who do not have a feedback loop in place.
* Provide a GDPR-compliant option for a complaint feedback loop.

1.1. Scope of this Experiment

The CFBL-Address header and the CFBL-Feedback-ID header are an experiment. Participation in this experiment consists of adding the CFBL-Address-Header on email sender side or by using the CFBL-Address-Header to send FBL reports to the provided address on mailbox provider side. Feedback on the results of this experiment can be emailed to the author or raised as an issue at https://github.com/jpbede/rfc-cfbl-address-header/.

The goal of this experiment is to answer the following questions based on real-world deployments:

* Is there interest among email sender and mailbox providers?
* If the mailbox provider adds this capability, will it be used by the senders?
* If the email sender adds this capability, will it be used by the mailbox provider?
* Does the presence of the CFBL-Address/CFBL-Feedback-ID field introduce additional security issues?

* What additional security measures/checks need to be performed at the mailbox provider before a complaint report is sent?

* What additional security measures/checks need to be performed at the email sender after a complaint report is received?

This experiment is considered successful if the CFBL-Address header has been implemented by multiple independent parties (email sender and mailbox provider) and these parties successfully use the address specified in the header to exchange feedback loop reports.

If this experiment is successful and these headers prove to be valuable and popular, it may be taken to the IETF for further discussion and revision. One possible outcome could be that a working group creates a specification for the standard track.

1.2. Difference to One-Click-Unsubscribe

For good reasons, the One-Click-Unsubscribe [RFC8058] signaling already exists, which may have several interests in common with this document. However, this header requires the List-Unsubscribe header, whose purpose is to provide the link to unsubscribe from a list. For this reason, this header is only used by operators of broadcast marketing lists or mailing lists, not in normal email traffic.

The main interest of this document now is to provide an automated way to signal mailbox providers an address for a complaint feedback loop. It is the obligation of the email sender to decide for themselves what action to take after receiving a notification; this is not the subject of this document.

2. Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The keyword "CFBL" in this document is the abbreviation for "complaint feedback loop" and will hereafter be used.

The keyword "MBP" in this document is the abbreviation for "mailbox provider", it is the party who receives an email, and will be used hereafter.
The keyword "email sender" in this document is used to describe the party who sends an email, this can be an MBP, a broadcast marketing list operator or any other email sending party. It will be used hereafter.

3. Requirements

3.1. Received email

This section describes the requirements that a received email, i.e. the email that is sent from the email sender to the MBP and about which a report is to be sent later, must meet.

3.1.1. Strict

If the domain in the [RFC5322].From and the domain in the CFBL-Address header are identical, this domain MUST be covered by a valid [DKIM] signature. In this case, the DKIM "d=" parameter and the [RFC5322].From field have identical DNS domains. This signature MUST meet the requirements described in Section 3.1.4.

The following example meets this case:

Return-Path: <sender@mailer.example.com>
From: Awesome Newsletter <newsletter@example.com>
To: receiver@example.org
Subject: Super awesome deals for you
CFBL-Address: fbl@example.com; report=arf
Message-ID: <a37e51bf-3050-2aab-1234-543a0828d14a@mailer.example.com>
Content-Type: text/plain; charset=utf-8
DKIM-Signature: v=1; a=rsa-sha256; d=example.com; s=news;
   h=Subject:From:To:Message-ID:CFBL-Feedback-ID:CFBL-Address;

This is a super awesome newsletter.

3.1.2. Relaxed

If the domain in CFBL-Address is a child domain of the [RFC5322].From, the [RFC5322].From domain MUST be covered by a valid [DKIM] signature. In this case, the DKIM "d=" parameter and the [RFC5322].From domain have a identical (Example 1) or parent (Example 2) DNS domains. This signature MUST meet the requirements described in Section 3.1.4.

Example 1:
Return-Path: <sender@mailer.example.com>
From: Awesome Newsletter <newsletter@mailer.example.com>
To: receiver@example.org
Subject: Super awesome deals for you
CFBL-Address: fbl@mailer.example.com; report=arf
Message-ID: <a37e51bf-3050-2aab-1234-543a0828d14a@mailer.example.com>
Content-Type: text/plain; charset=utf-8
DKIM-Signature: v=1; a=rsa-sha256; d=example.com;
  h=Content-Type:Subject:From:To:Message-ID:
  CFBL-Feedback-ID:CFBL-Address;

This is a super awesome newsletter.

Example 2:

Return-Path: <sender@mailer.example.com>
From: Awesome Newsletter <newsletter@mailer.example.com>
To: receiver@example.org
Subject: Super awesome deals for you
CFBL-Address: fbl@mailer.example.com; report=arf
Message-ID: <a37e51bf-3050-2aab-1234-543a0828d14a@mailer.example.com>
Content-Type: text/plain; charset=utf-8
DKIM-Signature: v=1; a=rsa-sha256; d=example.com;
  h=Content-Type:Subject:From:To:Message-ID:
  CFBL-Feedback-ID:CFBL-Address;

This is a super awesome newsletter.

3.1.3. Complex

If the domain in [RFC5322].From of differs from the domain in the
CFBL-Address header, the domain of the CFBL-Address header MUST be
covered by an additional valid [DKIM] signature. Both signatures
MUST meet the requirements described in Section 3.1.4.

This double DKIM signature ensures that both the domain owner of the
From: domain and the domain owner of the CFBL-Address: domain agree
to receive the complaint reports on the address from the CFBL-
Address: header.

The following example meets this case:
Return-Path: <sender@super-saas-mailer.com>
From: Awesome Newsletter <newsletter@example.com>
To: receiver@example.org
Subject: Super awesome deals for you
CFBL-Address: fbl@super-saas-mailer.com; report=arf
Message-ID: <a37e51bf-3050-2aab-1234-543a0828d14a@example.com>
Content-Type: text/plain; charset=utf-8
DKIM-Signature: v=1; a=rsa-sha256; d=example.com; s=news;
    h=Subject:From:To:Message-ID:CFBL-Feedback-ID:CFBL-Address;
    d=super-saas-mailer.com; s=system;
    h=Subject:From:To:Message-ID:CFBL-Feedback-ID:CFBL-Address;

This is a super awesome newsletter.

3.1.4.  DKIM signature

The CFBL-Address header MUST be included in the "h=" tag of the 
aforementioned valid DKIM-Signature. When the CFBL-Feedback-ID 
header is used, it MUST also be included in the "h=" tag of the 
aforementioned valid DKIM signature.

If the domain has neither the required coverage by a valid DKIM 
signature nor the required header coverage by the "h=" tag, the MBP 
SHALL NOT send a report email.

3.2.  Report email

The report email (sent by MBP to email sender) MUST have a valid 
[DKIM] signature. The aforementioned valid DKIM signature MUST cover 
the From: header [MAIL] domain, from which the report is sent to the 
email sender.

If the message does not have the required valid [DKIM] signature, the 
email sender SHALL NOT process this complaint report.

As part of this experiment, it is recommended to determine what 
plausibility and security checks are useful and achievable.

4.  Implementation

4.1.  Email senders

An email sender who wishes to receive complaints about their emails 
MUST include a CFBL-Address header in their messages.

The receiving complaint FBL address specified in the messages MUST 
accept [ARF] compatible reports by default. The email sender can 
OPTIONALLY request a [XARF] compatible report if they want one, as
described in Section 5.1. The MBP MAY send a [XARF] compatible report if it is technically possible for them to do so, otherwise a [ARF] compatible report will be sent.

It is strongly RECOMMENDED that these reports be processed automatically. Each sender must decide for themselves what action to take after receiving a report.

The email sender MUST take action to address the described requirements in Section 3.

4.2. Mailbox provider

If the MBP wants to process the complaints and forward it, they MUST query the CFBL-Address header and forward the report to the complaint FBL address.

By default, an [ARF] compatible report MUST be sent. Per [RFC6449] Section 3.2, a complaint report MUST be sent when a manual action has been taken e.g., when a receiver marks a mail as spam, by clicking the "This is spam"-button in any web portal or by moving a mail to junk folder, this also includes [IMAP] and [POP3] movements. The MBP SHALL NOT send any report when an automatic decisions has been made e.g., spam filtering.

The MBP MUST send a [XARF] compatible report when the email sender requests it as described in Section 5.1. If it is not possible for the MBP to send a [XARF] compatible report as requested, a [ARF] compatible report MUST be sent.

The MBP MUST validate and take action to address the described requirements in Section 3.

5. Complaint report

The complaint report MAY be a [XARF] report if the email sender requests it, and it is technically possible for the MBP to do so, otherwise the complaint report MUST be a [ARF] report.

The report MUST contain at least the Message-ID [MAIL]. If present, the header "CFBL-Feedback-ID" of the complaining email MUST be added additionally.

The MBP MAY omit or redact, as described in [RFC6590], all further headers and/or body to comply with any data-regulation laws.
It is highly RECOMMENDED that, if used, the CFBL-Feedback-ID includes a hard to forge component such as an [HMAC] using a secret key, instead of a plain-text string.

5.1. XARF compatible report

A email sender wishing to receive a [XARF] compliant report MUST append "report=xarf" to the Section 6.1. The resulting header would look like the following:

CFBL-Address: fbl@example.com; report=xarf

6. Header Syntax

6.1. CFBL-Address

The following ABNF imports fields, WSP, CRLF and addr-spec from [MAIL].

fields /= cfbl-address

cfbl-address = "CFBL-Address:" 0*1WSP addr-spec
               [";" 0*1WSP report-format] CRLF

report-format = "report=" ("arf" / "xarf")

6.2. CFBL-Feedback-ID

The following ABNF imports fields, WSP, CRLF and atext from [MAIL].

fields /= cfbl-feedback-id

cfbl-feedback-id = "CFBL-Feedback-ID:" 0*1WSP fid CRLF

fid = 1*(atext / ":")

7. Security Considerations

This section discusses possible security issues, and their possible solutions, of a complaint FBL address header.

7.1. Attacks on the FBL address

Like any other email address, a complaint FBL address can be an attack vector for malicious emails. For example, complaint FBL addresses can be flooded with spam. This is an existing problem with any existing email address and is not created by this document.
The email sender must take appropriate measures. One possible countermeasure would be a rate limit for the delivering IP. However, this should be done with caution; normal FBL email traffic must not be affected.

7.2. Automatic suspension of an account

Sending an FBL report against a mailbox can cause the account holder to be unreachable if an automatic account suspension occurs too quickly. An example: someone sends an invitation to his friends. For some reason, someone marks this mail as spam. Now, if there is too fast automatic account suspension, the sender's account will be blocked and the sender will not be able to access his mails.

MBPs and email senders must take appropriate measures to prevent this. MBPs and email senders therefore have, mostly proprietary, ways to assess the trustworthiness of an account. For example, MBPs and email senders may take into account the age of the account and/or any previous account suspension before suspending an account.

7.3. Enumeration attacks / provoking unsubscription

A malicious person may send a series of spoofed ARF messages to known complaint FBL addresses and attempt to guess a Message-ID/CFBL-Feedback-ID or any other identifiers. The malicious person may attempt to mass unsubscribe/suspend if such an automated system is in place. This is also an existing problem with the current FBL implementation and/or One-Click Unsubscription [RFC8058].

The sender of the received email must take appropriate measures. As a countermeasure, it is recommended that the CFBL-Feedback-ID, if used, use a hard-to-fake component such as a [HMAC] with a secret key instead of a plaintext string to make an enumeration attack impossible.

If it is impossible for the email sender to use a component that is difficult to fake, they should take steps to avoid enumeration attacks.

7.4. GDPR and other data-regulation laws

The provision of such a header itself does not pose a data protection issue. The resulting ARF report sent by the MBP to the email sender may violate a data protection law because it may contain personal data.
This document already addresses some parts of this problem and describes a privacy-safe way to send a FBL report. As described in Section 5, the MBP can omit the entire body and/or header and send only the required fields. As described in [RFC6590], the MBP can also redact the data in question. Nevertheless, each MBP must consider for itself whether this implementation is acceptable and complies with existing privacy laws.

As described in Section 5, it is also strongly RECOMMENDED that the Message-ID and, if used, the CFBL-Feedback-ID. contain a component that is difficult to forge, such as a [HMAC] that uses a secret key, rather than a plaintext string. See Section 9.3 for an example.

Using HMAC, or any other hard to forge component, ensures that only the email sender has knowledge about the data.

7.5. Abusing for Validity and Existence Queries

This mechanism could be abused to determine the validity and existence of an email address, which exhibits another potential privacy issue. Now, if the MBP has an automatic process to generate a complaint report for a received email, it may not be doing the mailbox owner any favors. As the MBP now generates an automatic complaint report for the received email, the MBP now proves to the email sender that this mailbox exists for sure, because it is based on a manual action of the mailbox owner.

The receiving MBP must take appropriate measures. One possible countermeasure could be, for example, pre-existing reputation data, usually proprietary data. Using this data, the MBP can assess the trustworthiness of an email sender and decide whether to send a complaint report based on this information.

8. IANA Considerations

8.1. CFBL-Address

The IANA is requested to register a new header field, per [RFC3864], into the "Provisional Message Header Field Names" registry:
Header field name: CFBL-Address
Applicable protocol: mail
Status: provisional
Author/Change controller: Jan-Philipp Benecke <jpb@cleverreach.com>
Specification document: this document

8.2. CFBL-Feedback-ID

The IANA is requested to register a new header field, per [RFC3864], into the "Provisional Message Header Field Names" registry:

Header field name: CFBL-Feedback-ID
Applicable protocol: mail
Status: provisional
Author/Change controller: Jan-Philipp Benecke <jpb@cleverreach.com>
Specification document: this document

9. Examples

For simplicity the DKIM header has been shortened, and some tags have been omitted.

9.1. Simple

Email about the report will be generated:

Return-Path: <sender@mailer.example.com>
From: Awesome Newsletter <newsletter@example.com>
To: me@example.net
Subject: Super awesome deals for you
CFBL-Address: fbl@example.com; report=arf
CFBL-Feedback-ID: 111:222:333:4444
Message-ID: <a37e51bf-3050-2aab-1234-543a0828d14a@mailer.example.com>
Content-Type: text/plain; charset=utf-8
DKIM-Signature: v=1; a=rsa-sha256; d=example.com; s=news;
   h=Subject:From:To:Message-ID:CFBL-Feedback-ID:CFBL-Address;

   Resulting ARF report:
9.2. GDPR safe report

Email about the report will be generated:

Return-Path: <sender@mailer.example.com>
From: Awesome Newsletter <newsletter@example.com>
To: me@example.net
Subject: Super awesome deals for you
CFBL-Address: fbl@example.com; report=arf
CFBL-Feedback-ID: 111:222:333:4444
Message-ID: <a37e51bf-3050-2aab-1234-543a0828d14a@mailer.example.com>
Content-Type: text/plain; charset=utf-8
DKIM-Signature: v=1; a=rsa-sha256; d=example.com; s=news;
   h=Subject:From:To:Message-ID:CFBL-Feedback-ID:CFBL-Address;

This is a super awesome newsletter.

Resulting ARF report contains only the CFBL-Feedback-ID:
9.3. GDPR safe report with HMAC

Email about the report will be generated:

Return-Path: <sender@mailer.example.com>
From: Awesome Newsletter <newsletter@example.com>
To: me@example.net
Subject: Super awesome deals for you
CFBL-Address: fbl@example.com; report=arf
CFBL-Feedback-ID: 3789e1ae1938aa2f0dfdfa48b20d8f8bc6c21ac34fc5023d63f9e64a43dfedc0
Message-ID: <a37e51bf-3050-2aab-1234-543a0828d14a@mailer.example.com>
Content-Type: text/plain; charset=utf-8
DKIM-Signature: v=1; a=rsa-sha256; d=example.com; s=news;
              h=Subject:From:To:Message-ID:CFBL-Feedback-ID:CFBL-Address;

This is a super awesome newsletter.

Resulting ARF report contains only the CFBL-Feedback-ID:
10. Acknowledgments

Technical and editorial reviews were provided by the colleagues at CleverReach, the colleagues at Certified Senders Alliance and eco.de, Levent Ulucan (Inxmail), Arne Allisat and Tobias Herkula (1&1 Mail & Media) and Sven Krohlas (BFK Edv-consulting).

11. References

11.1. Normative References


11.2. Informative References


Benecke Expires 5 September 2022 [Page 17]

Author’s Address

Jan-Philipp Benecke
CleverReach GmbH & Co. KG
Schafjueckenweg 2
26180 Rastede
Germany
Phone: +49 4402 97390-16
Email: jpb@cleverreach.com
Abstract

This document allows broader use of the Expires message header field for SMTP. Senders can then indicate when a message sent becomes valueless and can safely be deleted, while recipients would use the information to delete these valueless messages.

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 https://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 16 October 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. Introduction

The date and time of expiration can be used by the mailbox provider or the MUA to indicate to the user that certain messages could be deleted, in an attempt to unclutter the user’s mailbox and spare storage resources.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Header Field example

The field definition and syntax remain the same.

expires = "Expires" "":" date-time

Example:

Expires: Wed, 1 Dec 2021 17:22:57 +0000

Senders MUST NOT include more than one Expires header in the message they send.

If there is more than one Expires header then receivers SHOULD treat this as if no Expires header is present.
3. Security considerations

Dates in this header can be set a long way in the past or in the future, including outside the range of internal time representations in some programming environments - all software which processes the Expires header MUST be made safe against this possibility.

4. Advice to Senders

Senders SHOULD add the header field along with a relevant date and time when they know that the content of the message has no value after a given point of time (e.g. Commercial newsletters --especially when including time-limited offers, Event announcements, Social notifications, Time-limited access codes ...).

In all other cases, senders SHOULD NOT set an Expires header.

5. Advice to Receivers (Mailbox providers, Webmails and MUAs)

The expiration of a message’s validity would logically lead to the deletion of the message. However, users on most systems do not expect their emails to disappear, and may not be aware that any particular email has an Expires header. Therefore, no email should be silently and automatically deleted solely based on the value of the Expires header field.

Mailbox providers SHOULD explain to users how the information provided in the Expires header are processed, SHOULD indicate when viewing an expired message, and SHOULD give users control over the actions to take for expired messages.

The information provided in the header should be used as a signal that could be used to provide a feature or improved experience to the end-user. For instance, systems may allow users to set up an automatic rule to clean up expired email from specific senders or with specific characteristics, or provide a mode to quickly view and process all expired email.

In certain cases, email messages can be used as proof or element of investigation. As an early deletion may compromise the intended investigation, mailbox providers can ignore the Expires information in such cases.

Presence of the Expires header field MUST NOT be interpreted as a sign of legitimacy.
6. Past History of the Expires: header

[RFC4021] defines a number of header fields that can be added to Internet messages such as those used for mapping between X.400 and RFC822/MIME [RFC2156]. One of them is the Expires header field that provides the date and time at which a message is considered to lose its validity.

The same principle can be applied to the Expires header field in a SMTP context, whether the message comes from a X.400 gateway as initially intended in [RFC2156], or from a RFC821/SMTP MTA.

7. Acknowledgements

This document was informed by discussions with and/or contributions from Jonathan Loriaux, Charles Sauthier and Simon Bressier.

8. IANA Considerations

IANA is requested to update an existing entry in the Permanent Message Headers Field Names registry (https://www.iana.org/assignments/message-headers/message-headers.xhtml)

Header field name: Expires

Applicable protocol: mail

Status: standard

Author/Change controller: IETF

Specification document: this document

9. Normative References


10. Informative References


Authors’ Addresses

Benjamin Billon
Splio
Email: bbillon@splio.com

John Levine
Standcore LLC
Email: standards@standcore.com
A well-known URI for publishing ECHConfigList values.
draft-farrell-tls-wkesni-02

Abstract

We propose use of a well-known URI at which web servers can publish ECHConfigList values as a way to help get those published in the DNS.

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 https://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 3 June 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. Introduction

Encrypted ClientHello (ECH) [I-D.ietf-tls-esni] for TLS1.3 [RFC8446] defines a confidentiality mechanism for server names and other ClientHello content in TLS. For many applications, that requires publication of ECHConfigList data structures in the DNS. An ECHConfigList structure contains a list of ECHConfig values. Each ECHConfig value contains the public component of a key pair that will typically be periodically (re-)generated by a web server. Many web infrastructures will have an API that can be used to dynamically update the DNS RR values containing ECHConfigList values. Some deployments however, will not, so web deployments could benefit from a mechanism to use in such cases.

We define such a mechanism here. Note that this is not intended for universal deployment, but rather for cases where the web server doesn’t have write access to the relevant zone file (or equivalent). That zone file will eventually include an HTTPS or SVCB RR [I-D.ietf-dnsop-svcb-https] containing an ECHConfigList.

We use the term "zone factory" for the entity that does have write access to the zone file. We assume the zone factory (ZF) can also make HTTPS requests to the web server with the ECH keys.

We propose use of a well-known URI [RFC8615] on the web server that allows ZF to poll for changes to ECHConfigList values. For example, if a web server generates new ECHConfigList values hourly and publishes those at the well-known URI, ZF can poll that URI. When ZF sees new values, it can check if those work, and if they do, then update the zone file and re-publish the zone.
[This idea could: a) wither on the vine, b) be published as it’s own RFC, or c) end up as a PR for [I-D.ietf-tls-esni]. There is no absolute need for this to be in the RFC that defines ECH, so (b) seems feasible if there's enough interest, hence this draft. The source for this is in https://github.com/sftcd/wkesni/ PRs are welcome there too.]

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Example use of the well-known URI for ECH

An example deployment could be as follows:

* Web server generates new ECHConfigList values hourly at N past the hour via a cronjob
* ECHConfigList values are "current" for an hour, published with a TTL of 1800, and remain usable for 3 hours from the time of generation
* Web server has a set of "backend" sites - the DNS name for each such site is here represented as $BACKEND, which will end up as an SNI value to be encrypted inside an ECH extension
* Web server has a "front-end" site ($FRONT), where $FRONT will typically be the DNS name used in the ECHConfigList public_name field for ECHConfig version 0xff0d
* A cronjob creates a JSON file for each backend site at https://$FRONT/.well-known/ech/$BACKEND.json
* Each JSON file contains an array with the ECHConfigList values for that particular $BACKEND as shown in Figure 1 - the values in Figure 1 with ellipses are the values we want to eventually see in the DNS
* On the zone factory, a cronjob runs at N+3 past the hour, it knows all the names involved and checks to see if the content at those well-known URIs has changed or not
* If the content has changed the cronjob attempts to use the ECHConfigList values, and for each $BACKEND where that works, it updates the zone file and re-publishes the zone containing only the new ECHConfigList values
4. The ech well-known URI

When a web server ($\text{FRONT}$) wants to publish ECHConfigList information for a backend site ($\text{BACKEND}$) then it provides the JSON content defined in Section 5 at: https://$\text{FRONT}$/well-known/ech/$\text{BACKEND}$.json

The well-known URI defined here MUST be an https URL and therefore the zone factory verifies the correct $\text{FRONT}$ is being accessed. If there is any failure in accessing the well-known URI, then the zone factory MUST NOT modify the zone.

5. The JSON structure for ECHConfigList values

[[Since the specifics of the JSON structure in Figure 1 are very likely to change, this is mostly TBD. What is here for now, is what the author has currently implemented simply because it worked ok and was easy to do:-)]

[[Might change this due to retry-configs and now that I’ve implemented split-mode]]

```
[
  {
    "desired-ttl": 1800,
    "ports": [ 443, 8413 ],
    "echconfiglist": "AD7+DQA65wAgAC..AA=="
  },
  {
    "desired-ttl": 1800,
    "ports": [ 443, 8413 ],
    "echconfiglist": "AD7+DQA65wAgAC..AA=="
  }
]
```

Figure 1: Sample JSON

The JSON file at the well-known URI MUST contain an array with one or more elements. Each element of the array MUST have these fields:

- desired-ttl: contains a number indicating the TTL that the web server would like to see used for this RR. The zone factory MUST NOT use a longer TTL.
- ports: this has a list of the TCP ports on which the web server with the relevant key pair will listen (needed to produce the correct zone file).
- ECHConfigList: contains the value to be used as a base64 encoded string.
The JSON file contains an array for a couple of reasons:

* As TLS authentication doesn’t really distinguish ports, servers on the same host could in any case cheat on one another, so we may as well just read one JSON file per name.
* Different ports could map to different sets of ECHConfig values
* As ECHConfigList is (regrettably:-) an extensible structure, it may be necessary to publish different ECHConfigList values to get best interoperability.

6. Zone factory behaviour

The zone factory SHOULD check that the presented ECHConfigList values work with the $BACKEND server before publication. A "special" TLS client may be needed for this check, that does not require the ECHConfigList value to have already been published in the DNS. [[I guess that calls for the zone factory to know of a "safe" URL on $BACKEND to try, or maybe it could use HTTP HEAD? Figuring that out is TBD. The ZF could also try a GREASEd ECH and see if the retry-configs it gets back is one of the ECHConfig values in the ECHConfigList.]]

A careful zone factory could explode the ECHConfigList value presented into "singleton" values with one public key in each and test each for each port claimed.

The zone factory SHOULD publish all the ECHConfigList values that are presented in the JSON file, and that pass the check above.

The zone factory SHOULD only publish ECHConfigList values that are in the latest version of the JSON file. This leaves the control of "expiry" with the web server, so long as the ECHConfigList values presented actually work. [[An alternative could be to have the new values just be appended to the zone, but that’d require some form of "notAfter" value in the JSON file which seems unnecessary and more complex.]]

The SCVB and HTTPS RR specification [I-D.ietf-dnsop-svcb-https] defines how and where the ECHConfigList values for $BACKEND needs to be published in the DNS. The zone factory is assumed to be in control of how ECHConfigList values are included in such RRs.
A possibly interesting (unintended) consequence of this design is that once a TLS client has first gotten an ECHConfigList from the DNS for $BACKEND with the ECHConfigList structure containing the public_name field, the TLS client would know both $FRONT and $BACKEND and so could later probe for this .well-known as an alternative to doing so via DoT/DoH. Probably not something a web browser might do, but could be fun for other applications maybe.

[[The extent to which retry-configs could be used for a similar purpose might be worth considering. But the JSON stuff here may still be needed if implementations (such as mine:-) tend to only return one ECHConfig in retry-configs.]]

7. Security Considerations

This document defines another way to publish ECHConfigList values. If the wrong keys were read from here and published in the DNS, then clients using ECH would do the wrong thing, likely resulting in denial of service, or a privacy leak, or worse, when TLS clients attempt to use ECH with a backend web site. So: Don’t do that:-)

8. Acknowledgements

Thanks to Niall O’Reilly for a quick review of -00.

9. IANA Considerations

[[TBD: IANA registration of a .well-known. Also TBD - how to handle I18N for $FRONT and $BACKEND within such a URL.]]

10. Normative References


[I-D.ietf-tls-esni]

[I-D.ietf-dnsop-svcb-https]

Appendix A. Change Log

[[RFC editor: please remove this before publication.]]

From -01 to -02:
* General changes from ESNI to ECH.

From -00 to -01:
* Re-structured a bit after re-reading rfc8615

Author’s Address

Stephen Farrell
Trinity College Dublin
Dublin
2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie
Open Ethics Transparency Protocol

draft-lukianets-open-ethics-transparency-protocol-00

Abstract

The Open Ethics Transparency Protocol (OETP) is an application-level protocol for publishing and accessing ethical Disclosures of IT Products and their Components. The Protocol is based on HTTP exchange of information about the ethical "postures", provided in an open and standardized format. The scope of the Protocol covers Disclosures for systems such as Software as a Service (SaaS) Applications, Software Applications, Software Components, Application Programming Interfaces (API), Automated Decision-Making (ADM) systems, and systems using Artificial Intelligence (AI). OETP aims to bring more transparent, predictable, and safe environments for the end-users. The OETP Disclosure Format is an extensible JSON-based format.

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 https://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 28 May 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.
# Table of Contents

1. Introduction ......................................................... 3  
2. Requirement Levels .................................................. 4  
3. Terminology ........................................................... 4  
4. Protocol Model ......................................................... 5  
   4.1. Creation of Disclosure ......................................... 6  
   4.1.1. Cryptographic Signature .................................... 6  
   4.1.2. Immutable Storage .......................................... 6  
   4.1.3. Visual Labeling ............................................ 6  
   4.1.4. Voice Labeling ............................................. 6  
   4.2. Access to Disclosure ........................................... 6  
   4.2.1. Initial Request to a Disclosure file ...................... 6  
   4.2.2. Access to Visual Trust Labels .............................. 7  
   4.2.3. Requirements for placement of Integrity Signature in  
          Visual Label ............................................... 7  
   4.2.4. Conformity assessment marks ............................... 7  
   4.2.5. Accessibility considerations ............................... 7  
4.3. Verification and Validation of Disclosure ....................... 7  
   4.3.1. Automated Disclosure processing ............................ 7  
   4.3.2. Validation of Vendor’s Disclosures ........................ 8  
   4.3.3. Verification of Vendor’s Disclosures ........................ 8  
   4.3.4. Progressive Verification .................................. 8  
4.4. End-to-end transparency and formation of the composite  
     Disclosure ................................................................ 8  
   4.4.1. Open Supplier Policy ....................................... 8  
   4.4.2. Request for Supplier’s Disclosures ........................ 10  
   4.4.3. Disclosure Chaining ......................................... 10  
   4.4.4. Generation of the Composite Disclosure ................. 10  
5. Example OETP Disclosure File ......................................... 11  
6. Security Considerations ................................................ 12  
   6.1. Response content ................................................. 12  
   6.2. Spoofing ............................................................ 13  
   6.3. Falsification ..................................................... 13  
7. IANA Considerations ................................................... 13  
8. Areas for Future Study ................................................. 13  
9. References ............................................................... 14  
   9.1. Normative References .......................................... 14  
   9.2. Informative References ....................................... 14  
Appendix A. Appendix ........................................................ 14  

Lukianets                   Expires 28 May 2022                   [Page 2]
1. Introduction

The Open Ethics Transparency Protocol (OETP or Protocol) describes the creation and exchange of voluntary ethics Disclosures for IT products. It is brought as a solution to increase the transparency of how IT products are built and deployed. This document provides details on how disclosures for data collection and data processing practice are formed, stored, validated, and exchanged in a standardized and open format.

OETP provides facilities for:

* *Informed consumer choices* : End-users able to make informed choices based on their own ethical preferences and product disclosure.

* *Industrial-scale monitoring* : Discovery of best and worst practices within market verticals, technology stacks, and product value offerings.

* *Legally-agnostic guidelines* : Suggestions for developers and product-owners, formulated in factual language, which are legally-agnostic and could be easily transformed into product requirements and safeguards.

* *Iterative improvement* : Digital products, specifically, the ones powered by artificial intelligence could receive nearly real-time feedback on how their performance and ethical posture could be improved to cover security, privacy, diversity, fairness, power balance, non-discrimination, and other requirements.

* *Labeling and certification* : Mapping to existing and future regulatory initiatives and standards.

The Open Ethics Transparency Protocol (OETP) is an application-level protocol for publishing and accessing ethical Disclosures of IT products and their components. The Protocol is based on HTTP exchange of information about the ethical "postures", provided in an open and standardized format. The scope of the Protocol covers
Disclosures for systems such as Software as a Service (SaaS) Applications, Software Applications, Software Components, Application Programming Interfaces (API), Automated Decision-Making (ADM) systems, and systems using Artificial Intelligence (AI). OETP aims to bring more transparent, predictable, and safe environments for the end-users. The OETP Disclosure Format is an extensible JSON-based format.

2. Requirement Levels

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

Disclosure: Disclosure (Ethics Disclosure, or self-disclosure) is application-specific information about the data collection, data-processing, and decision-making practices of a Product, provided by the Product Vendor (an individual developer or an organization).

Disclosure Feed: A historical sequence of Disclosures, made for a specific Product.

Vendor: A legal person (an individual developer or an organization) that owns one or several end-user Products, or acts as a Supplier and provides Components for other Vendors.

Integrator: A legal person (an individual developer or an organization) that deploys technology-powered services to the end-users based on Product(s) from third-party Vendors.

Product: An IT system in the form of software, software as a service system, application, software component, application programming interface, or a physically embodied automated decision-making agent.

Component: An IT system supplied by Vendor and integrated/embedded into end-user Products. Components themselves do not necessarily interface with end-users.

Upstream Component: A Component that sends its outputs to the Product Downstream in the data processing chain. Disclosure for the Upstream Component is represented as a Child relative to the Disclosure node of the Downstream Product.
Downstream Component: A Component that receives inputs from the Components Upstream in the data processing chain. Disclosure for the Downstream Component is represented as a Parent relative to the Disclosure node of the Upstream Component.

Automated Decision-Making (ADM): Automated decision-making is the process of making a decision by automated means without any human involvement. These decisions can be based on factual data, as well as on digitally created profiles or inferred data.

OETP Disclosure Format: A machine-readable Disclosure with predefined structure, supplied in the JSON format.

Validation: A sequence of automated software-based checks to control validity and security elements in the OETP Disclosure.

Auditor: A third-party legal person trusted to perform Verification checks and to issue Verification Proofs.

Auditing software: An automated software-based tool authorized to perform Verification checks and to issue Verification Proofs.

Verification: A procedure to control the correspondence of the elements in the OETP Disclosure and the actual data processing and data collection practices of the Vendors.

Verification Proof: A result of the formal Disclosure Verification procedure presented to a requestor.

Chaining: A process of combining Disclosures of individual Components into a composite high-level Disclosure for a Product.

Label: User-facing graphical illustrations and textual descriptions of the Product that facilitate understanding of the values and risks the Product carries.

4. Protocol Model

The Disclosure creation and delivery consist of the two parts, starting from (I) the submission of the Disclosure form, chaining of the Suppliers’ Disclosures, Signature of the disclosed information, and to the delivery part (II) that first checks that the Disclosure is Valid, and then that the information specified in it is Verified by the third-parties. Figure 2 shows disclosure creation steps.
4.1. Creation of Disclosure

The initial Disclosure is created by filling a standardized disclosure form (for example, see 1. https://openethics.ai/label/ (https://openethics.ai/label/)). A Vendor representative, a Product Owner, or a Developer, MUST submit data-processing and data-collection information about the Product. The information about the end-point URL, as well as a contact email address, MUST be specified. Disclosure MAY also be created in a fully automated way as a part of the CI/CD DevOps pipeline. Figure 3 shows basic disclosure submission process.

4.1.1. Cryptographic Signature

The Disclosure is organized into a predefined data schema and MUST be cryptographically signed by the Signature Generator (Open Ethics or federated providers) using standard SHA3-512 hash implementation. The integrity hash MUST be appended to a disclosure as the OETP.schema.integrity element.

4.1.2. Immutable Storage

Both the signature integrity hash and the Disclosure SHOULD be stored in the log-centric root database and MAY be mirrored by other distributed databases for redundancy and safety.

4.1.3. Visual Labeling

Open Ethics Label SHOULD be automatically generated by mirroring the submitted Disclosure into a set of graphical icons and simple human-readable descriptions. Additional Labels MAY be generated following successful third-party Verification and by mapping the regulatory requirements to Verified Disclosures.

4.2. Access to Disclosure

4.2.1. Initial Request to a Disclosure file

The most recent OETP file SHOULD be stored in the root of the Product’s specified end-point URL, allowing requests to the OETP file from third-party domains. When establishing a Vendor relationship, the Integrator or a downstream Vendor MAY examine the Disclosure for their Components using the following HTTP request: GET https://testexample.com/oetp.json, where _testexample.com_ is the URL of the Supplier’s end-point.
4.2.2. Access to Visual Trust Labels

A Vendor SHOULD place a visual Label generated as a result of the Disclosure process in the Product informational materials (for example Marketing Materials, User Guides, Safety Instructions, Privacy Policy, Terms of Service, etc). The Label reflects the content of the Disclosure and SHOULD be displayed in any digital media by embedding a software widget. Visual labels in the print media SHOULD carry a visually distinguishable Integrity signature to enable manual Validation by the User.

4.2.3. Requirements for placement of Integrity Signature in Visual Label

* *Labels in the online digital media* MUST be generated automatically based on the content of the Disclosure and MUST contain a URL allowing to check the complete Integrity hash and explore more detailed information about the Disclosure.

* *Labels in the offline media* MUST be generated automatically based on the content of the Disclosure and should carry the first 10 digits of the corresponding Integrity hash.

4.2.4. Conformity assessment marks

Based on the Verification performed for the OETP Disclosure file, the labels MAY include Conformity assessment marks, Certification marks, as well as marks showing adherence to certain standards. These marks MAY be generated and displayed automatically based on the Verification Proofs.

4.2.5. Accessibility considerations

Accessibility of the Labels for the visually impaired Users SHOULD be considered. The OETP Processing system MUST provide alternative forms of the Label so that text-to-speech tools could be used to narrate the Label.

4.3. Verification and Validation of Disclosure

4.3.1. Automated Disclosure processing

The automated Disclosure processing is enabled by requests to both the Open Ethics Disclosure database powered by Disclosure Identity Providers and the Product’s OETP Disclosure file.
4.3.2. Validation of Vendor’s Disclosures

The OETP Processing system MUST compare integrity hashes in the Open Ethics Disclosure database and entries that arrive as a result of the Disclosure Request response.

4.3.3. Verification of Vendor’s Disclosures

Every disclosure SHOULD be checked for the existence of the external Verification from Auditors for the entire Disclosures or one of Disclosure elements.

4.3.4. Progressive Verification

To raise a level of trust to a Disclosure, a Vendor MAY decide to opt-in for a third-party Disclosure Verification. OETP suggests a Progressive Verification scheme where multiple independent external Verification Proofs COULD be issued by third parties to confirm the information specified in the Disclosure.

The Progressive Verification applies to a whole Disclosure, or to specific elements of the Disclosure.

Figure 4 displays a general scheme for Disclosure requests and responses.

The following elements MAY serve as sources for various kinds of Verification proofs: * Qualified Auditor reports * Qualified Vendor of Auditing software tests * Certification Authority assessments * Conformity assessments * User Feedback * Market Brokers * Real-time Loggers

4.4. End-to-end transparency and formation of the composite Disclosure

IT industry is getting more mature with Vendors becoming more specialized. Surface-level transparency is not sufficient as supply chains are becoming more complex and distributed across various Components. The following steps MUST be satisfied for the end-to-end transparency:

4.4.1. Open Supplier Policy

Every Integrator or a Vendor SHOULD disclose the information about their Suppliers (sub-processing Vendors), indicating the scope of the data processing in the Components they provide.
If the Supplier information is not provided, Disclosure SHOULD contain information that a Vendor (Integrator) has not provided Supplier information.

4.4.1.1. First-party Components

For greater transparency, Vendors may decide to reveal Components even if they originate from themselves (first-party Components). For the first-party Component, the Supplier identity information SHOULD NOT be provided because it was already disclosed earlier.

Required: (Section 4.4.1.3.2) only

4.4.1.2. Third-party Components

When disclosing Components originating from the third-party Vendors SHOULD provide both the Supplier identity information and Component information

Required: (Section 4.4.1.3.1, Section 4.4.1.3.2)

4.4.1.3. Elements of Supplier disclosure

4.4.1.3.1. Supplier identity

* Vendor Name
* Vendor URL
* Vendor ID
* Vendor DPO Contact Email

4.4.1.3.2. Component information

* Component Scope of use
* Personal Data Being Processed by Component
* Is a Safety Component (YES)/(NO)
* Component URL (if different from the Vendor URL)
* Component Disclosure URL (if different from the default Component URL/oetp.json)
* Component DPO Contact (if different from Vendor DPO Contact Email)
4.4.2. Request for Supplier’s Disclosures

The OETP Processing system MUST send GET requests to the URLs of each Component to obtain their Disclosures. Based on the response to each Disclosure request, the OETP Processing system MUST specify which Components have Disclosures and which don’t have Disclosures.

Figure 5 shows the process of how Disclosure Chaining requests and responses happen.

4.4.3. Disclosure Chaining

The same Request-response operation applies recursively for Components of the Components, and for the Components of the Components of the Components, etc. It is proposed to view the supply chain as a tree-like hierarchical data structure, where the information about Components is assembled using Level Order Tree Traversal algorithm.

In this tree: * Node is a structure that contains Component’s Disclosure; * Root is the top Node representing a Product’s Disclosure information; * Edge is the connection between one Node and another, representing the scope of the Data Processing by the Component.

Figure 6 displays the order of the Disclosure Chaining with Level Order Tree Traversal algorithm.

4.4.4. Generation of the Composite Disclosure

The current consensus from the user & developer community suggests that Composite Disclosure should follow The "Weakest Link" model. According to this model, the risk that the Product is carrying should not be considered any less than the risk for each of the Components.

Formally this approach could be illustrated with the use of a conjunction table for risk modeling (see Table 1). The Truth Table for Logical AND operator below takes one risk factor and evaluates risk outcomes as High (H) or Low (L) for hypothetical Disclosure options of the Product(P) and its Component(C).
<table>
<thead>
<tr>
<th>Disclosed risk of P</th>
<th>Disclosed risk of C</th>
<th>Composite P &amp; C</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td><em>L</em></td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td><em>H</em></td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td><em>H</em></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td><em>H</em></td>
</tr>
</tbody>
</table>

Table 1: Conjunction Table for Risk Modeling

Further evaluation of this approach is required.

5. Example OETP Disclosure File
Figure 1: Example OETP Disclosure File

6. Security Considerations

6.1. Response content

OETP exchanges data using JSON [RFC7159] which is a lightweight data-interchange format. A JSON-based application can be attacked in multiple ways such as sending data in an improper format or embedding attack vectors in the data. It is important for any application using JSON format to validate the inputs before being processed. To mitigate this attack type, the JSON Key Profile is provided for OETP responses.
6.2. Spoofing

OETP Processors should be aware of the potential for spoofing attacks where the attacker publishes an OETP disclosure with the OETP.snapshot value from another product, or, perhaps with an outdated OETP.snapshot.label element. For example, an OETP Processor could suppress the display of falsified entries by comparing the snapshot integrity from the submission database and a calculated hash for the OETP.snapshot object. In that situation, the OETP Processor might also take steps to determine whether the disclosures originated from the same publisher require further investigation of the Disclosure Feed and alert the downstream OETP Processors.

6.3. Falsification

Dishonest or falsified Disclosures is a problem that is hard to address generally. The approach to it is public control and systematic checks. Vendors or user-facing applications and services could further raise the level of trust in their Disclosures by implementing programmatic control scoring mechanisms, as well as the external verification by trusted Auditors.

7. IANA Considerations

This document has no IANA actions.

8. Areas for Future Study

The following topics not addressed in this version of document are possible areas for the future study:

* IANA requests for the Data Processor identity management.
* Extensibility of the OETP Disclosure Format.
* Evaluate other methods of Generation of the Composite Disclosure based on the Disclosure Tree
* Disclosure Chaining mechanisms and various use-cases.
* Typical scenarios and templates for Disclosure submissions.
* Mapping of the regulatory requirements and future Disclosure elements.
* Standardizing Privacy Disclosure and PII data-collection practices.
* Enhancing Label accessibility with ARIA W3C Recommendation and other approaches

* Use of the OETP Disclosure in the ADM explainability (XAI).

9. References

9.1. Normative References


9.2. Informative References


Appendix A. Appendix

A.1. Figures

Diagrams could be built from code using below *.puml files automatically using PlantUML (https://plantuml.com/).

A.1.1. Creation of Disclosure
@startuml

title Disclosure Creation Process

skinparam roundCorner 15

actor "Supplier A" as SA
actor "Supplier B" as SB
actor Vendor as V

cOMPONENT "Component A" as CA
COMPONENT "Component B" as CB
FILE "Disclosure A" as DA
FILE "Disclosure B" as DB
FILE "Composite Disclosure" as D

V-right->(Creation):disclose
SA-up->CA
SB-up->CB
CA-up->DA
CB-up->DB
DA-up->(Chaining)
DB-up->(Chaining)
(Creation)->(Chaining)
(Chaining)->(Validation)
(Validation)->(Verification)
(Verification)->D
@enduml

Figure 2: Creation of the Disclosure

A.1.2. Basic Disclosure Submission
@startuml

title Basic Disclosure Submission

skinparam roundCorner 15
autonumber

actor Vendor
database "Disclosure Identity Provider" as ID
control "Signature Generator" as SIG
database "Federated Identity Provider" as DIS

Vendor -> ID: Request with Disclosure payload
ID -> ID: Validate input
ID -> SIG: Structured Data, Initialized

ID <= SIG: SHA3-512 integrity hash
group Distributed Identity Storage
DIS <= SIG: SHA3-512 integrity hash

end

ID -> ID: Log OETP file and a corresponding integrity hash
Vendor <= ID: OETP Disclosure File
@enduml

Figure 3: Basic Disclosure Submission

A.1.3. Progressive Verification Scheme for Disclosures
Figure 4: Progressive Verification Scheme for Disclosures

A.1.4. Disclosure Chaining: Request-Response
@startuml
title Disclosure Chaining: Request-Response

start
repeat
:Request Component’s Disclosure;
if (Disclosure Obtained?) then (yes)
  :Validate Disclosure;
  :Verify Disclosure;
  :Chain Disclosure;
  :Obtain list of Child Components;
if (Supplier information exists?) then (yes)
  :Update Tree with (yet) Unchained Disclosures;
else (no)
  #Gold:**Alert** "Vendor has not provided Supplier information";
endif
else (no)
  #pink:**Alert** "Vendor has not provided any Disclosure";
stop
endif
repeat while (Unchained Disclosures in the Disclosure Tree?) is (yes) not (no)
  **Generate** Composite Disclosure;
  #palegreen:**Display** Label for "Composite Disclosure";
stop
@enduml

Figure 5: Disclosure Chaining: Request-Response

A.1.5. Disclosure Chaining: Level Order Traversal
Figure 6: Disclosure Chaining: Level Order Traversal

Acknowledgments

Part of this work related to Verification and Validation of Disclosure and Disclosure Chaining was supported by the H2020 Programme of the European Commission under Article 15 of Grant Agreement No. 951972 StandICT.eu 2023

The Open Ethics community and expert volunteers contributed with their valuable feedback, discussions, and comments. Thank you Ashley Duque Kienzle, Angela Kim, Ioannis Zempekakis, Karl Muedespacher, Ida Varosanec, Claudia Del Pozo, Joerg Buss, Mariia Kriuchok, Minhaaj Rehman, Oleksii Molchanovskyi, Roberta Barone, Phil Volkofsky and others.

Author’s Address

Nikita Lukainets
Open Ethics

Email: n.lukianets@openethics.ai
Inside MLS Message Interop (IMMI) instant message content
draft-mahy-dispatch-immi-content-00

Abstract

This document defines a profile intended for instant messaging interoperability of messages end-to-end encrypted inside the MLS (Message Layer Security) Protocol. It adapts prior work (CPIM) to work well in the MLS context.

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 https://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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. 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 RFC 2119 [RFC2219].

The terms MLS client, MLS group, and KeyPackage have the same meanings as in the MLS protocol [I-D.ietf-mls-protocol].
2. Introduction

MLS [I-D.ietf-mls-protocol] is a group key establishment protocol motivated by the desire for group chat with efficient end-to-end encryption. While one of the motivations of MLS is interoperable standards-based secure messaging, the MLS protocol does not define or prescribe any format for the encrypted "application messages" encoded by MLS. The development of MLS was strongly motivated by the needs of a number of Instant Messaging (IM) systems, which encrypt messages end-to-end using variations of the Double Ratchet protocol [].

End-to-end encrypted instant messaging was also a motivator for the Common Protocol for Instant Messaging (CPIM) [RFC3862], however the model used at the time assumed standalone encryption of each message using a protocol such as S/MIME [RFC8551] or PGP [RFC3156] to interoperate between IM protocols such as SIP [RFC3261] and XMPP [RFC6120]. For a variety of practical reasons, interoperable end-to-end encryption between IM systems was never deployed commercially.

There are now several vendors prepared to implement MLS. In order to enable interoperable messaging conveyed "inside" MLS application messages, some additional specification and some minor changes are required. Also, the expectation of what constitutes basic features common across multiple IM systems has grown. It would be beneficial to provide an interoperable format for these additional features as well. Most of these features can be implemented using a profile which describes how to use already-defined URIs, message headers, and MIME types.

This proposal assumes that MLS clients can advertise MIME types they support and that MLS clients can determine what MIME types are required to join a specific MLS group. A companion proposal [I-D.mahy-dispatch-immi-mls-mime] defines two MLS extensions which meets this requirement. It would allow implementations to define groups with different MIME type requirements and it would allow MLS clients to send extended or proprietary messages that would be interpreted by some members of the group while assuring that an interoperable end-to-end encrypted baseline is available to all members, even when the group spans multiple systems or vendors.

Below is a list of some features commonly found in IM group chat systems:

* plain text and rich text messaging
* delivery notifications
* read receipts
* replies
* reactions
* edit or delete previously sent messages
* expiring messages
* knock / ping
* shared files/audio/videos
* calling / conferencing

3. Overview

3.1. Naming schemes

IM systems have a number of types of identifiers. Not all systems use every type:

* client/device identifier (internal representation)
* user identifier
* handle identifier (external, friendly representation)
* group conversation identifier
* group or or channel name (external, friendly representation)
* team identifier (less common)

One user may have multiple clients (for example a mobile and a desktop client). A handle may refer to a single user or it may redirect to multiple users. In some systems, the user identifier is a handle. In other systems the user identifier is an internal representation, for example a UUID. Handles may be changed/renamed, but hopefully internal user identifiers do not. Unqualified handles are often prefixed with a commercial at-sign ("@").

Likewise, group conversation identifiers could be internal or external representations, whereas group names or channel names are often external friendly representations. Unqualified channel names are often prefixed with a hash character ("#"). Some systems have an additional level of hierarchy with a team identifier under which groups/channels can be organized and authorized.

This proposal relies on URIs for naming and identifiers. All the example use the im: URI scheme (defined in [RFC3862]), but any instant messaging scheme is acceptable.

3.2. Negotiation of MIME types

As most IM systems are proprietary, standalone systems, it is useful to allow clients to send and receive proprietary formats among themselves. Using the multipart/alternative MIME wrapper, clients can send a message using the basic functionality described in this document AND a proprietary format for same-vendor clients simultaneously over the same group with end-to-end encryption.
Internet-Draft Inside MLS IM content March 2022

[I-D.mahy-dispatch-immi-mls-mime] contains the actual MLS extensions useful for negotiating MIME types. The profile in this document requires support for receiving message/cpim, text/plain, text/markdown, and multipart MIME. All other mime types (including some recommended in this profile) are optional.

Example sending this profile and proprietary messaging protocol simultaneously.

Content-type: multipart/alternative

3.3. CPIM and MIME headers

We assume that an MLS group is already established and that either out-of-band or using the MLS protocol or MLS extensions that the following is known to every member of the group:

* The membership of the group (via MLS).
* The identity of any MLS client which sends an application message (via MLS).
* The MLS group ID (via MLS)
* The human readable name(s) of the MLS group, if any (out-of-band or extension).
* Which MIME types are mandatory to implement (proposed extension).
* For each member, the MIME types each supports (proposed extension).

For all messages the message header equivalent of To (the MLS group) and Sender fields (MLS sender) is already known and is therefore redundant. Every message contains a message/cpim header which includes the From, DateTime, and Message-ID fields. The From field contains the external, user-friendly representation of the Sender.

Messages sent to an MLS group are delivered to every member of the group active during the epoch in which the message was sent.

It is also mandatory to understand are the following MIME headers:

* Content-Type
* Content-Disposition
* Content-Length

4. Example
4.1. Original Message

In this example, Alice Smith sends a rich-text (Markdown) [RFC7763] message to the Engineering Team MLS group. The following values are implied as if headers were present:

* Implied Sender header from MLS sender:  im:3b52249d-68f9-45ce-8bf5-c799f3cad7ec-0003@example.com
* Implied To header from MLS group: "Engineering Team" im:9dc867ca-3a01-4385-bb69-1573601c3c0c@example.com

Content-type: message/cpim

From: <im:alice-smith@example.com>
DateTime: 2022-02-08T22:13:45-00:00
Message-ID: <28fd19857ad7@example.com>
Content-Type: text/markdown;charset=utf-8

Hi everyone, we just shipped release 2.0. __Good work__!

4.2. Reply

A reply message looks similar, but contains an In-Reply-To CPIM header with the ID of the original message. The implied To header is the same all example messages in this section. The implied Sender header is always the MLS sender, and will not be shown in subsequent example messages.

Content-type: message/cpim

From: <im:bob-jones@example.com>
DateTime: 2022-02-08T22:13:57-00:00
Message-ID: <e701beee59f9@example.com>
In-Reply-To:  <28fd19857ad7@example.com>
Content-Type: text/markdown;charset=utf-8

Right on! _Congratulations_ ’all!

4.3. Reaction

A reaction, uses the reaction Content-Disposition token defined in [RFC9078]. This Content-Disposition token indicates that the intended disposition of the contents of the message is a reaction.
The content in the sample message is a single Unicode heart character (U+2665). Discovering the range of characters each implementation could render as a reaction can occur out-of-band and is not within the scope of this proposal. However, an implementation which receives a reaction character string it does not recognize could render the reaction as a reply, possibly prefixing with a localized string such as "Reaction: ". Note that a reaction could theoretically even be another media type (ex: image, audio, or video), although not currently implemented in major instant messaging systems.

Content-type: message/cpim
From: <im:cathy-washington@example.com>
DateTime: 2022-02-08T22:13:57-00:00
Message-ID: <1a771ca1d84f@example.com>
In-Reply-To: <28fd19857ad7@example.com>

4.4. Mentions

In instant messaging systems and social media, a mention allows special formatting and behavior when a name, handle, or tag associated with a known group is encountered, often when prefixed with a commercial-at "@" character for mentions of users or a hash 
"#" character for groups or tags. A message which contains a mention may trigger distinct notifications on the IM client.

We can convey a mention by linking the user, handle, or tag URI in Markdown or HTML rich content. For example, a mention using Markdown is indicated below.

Content-type: message/cpim
From: <im:cathy-washington@example.com>
DateTime: 2022-02-08T22:14:03-00:00
Message-ID: <4dcab7711a77@example.com>

Kudos to [@Alice Smith](im:alice-smith@example.com) for making the release happen!
The same mention using HTML [W3C.CR-html52-20170808] is indicated below.

Content-type: message/cpim

From: <im:cathy-washington@example.com>
DateTime: 2022-02-08T22:14:03-00:00
Message-ID: <4dcab7711a77@example.com>

Content-Type: text/html;charset=utf-8

<p>Kudos to <a href="im:alice-smith@example.com">@Alice Smith</a> for making the release happen!</p>

4.5. Edit

Unlike with email messages, it is common in IM systems to allow the sender of a message to edit or delete the message after the fact. Typically the message is replaced in the user interface of the receivers (even after the original message is read) but shows a visual indication that it has been edited.

We reuse the Supersedes header from MIXER [RFC2156], because the semantics are correct: the message included in the body is a replacement for the message with the superseded message ID.

Here Bob Jones corrects a typo in his original message:

Content-type: message/cpim

From: <im:bob-jones@example.com>
DateTime: 2022-02-08T22:13:57-00:00
Message-ID: <89d3472622a4@example.com>
Supersedes: <e701beee59f9@example.com>

Content-Type: text/markdown;charset=utf-8

Right on! _Congratulations_ y’all!

4.6. Delete

In IM systems, a delete means that the author of a specific message has retracted the message, regardless if other users have read the message or not. Typically a placeholder remains in the user interface showing that a message was deleted. Replies which reference a deleted message typically hide the quoted portion and reflect that the original message was deleted.
If Bob deleted his message instead of modifying it, we would represent it using the Supersedes header with an empty body, as shown below.

```
Content-type: message/cpim

From: <im:bob-jones@example.com>
DateTime: 2022-02-08T22:13:57-00:00
Message-ID: <89d3472622a4@example.com>
Supersedes: <e701beee59f9@example.com>
Content-Length: 0
```

4.7. Expiring

Expiring messages are designed to be deleted automatically by the receiving client at a certain time whether they have been read or not. As with manually deleted messages, there is no guarantee that a uncooperative client or a determined user will not save the content of the message, however most clients respect the convention.

MIXER defines an Expires header which is also used sent simply by including an Expires header in the CPIM message body.

To avoid using two different date header syntaxes, we define an ExpiresDateTime header, which uses the same date/time format as CPIM’s DateTime header. The semantics of the header are that the message is automatically deleted by the receiving clients at the indicated time without user interaction or network connectivity necessary.

```
Content-type: message/cpim

From: <im:alice-smith@example.com>
DateTime: 2022-02-08T22:49:03-00:00
Message-ID: <5c95a4dfddab@example.com>
ExpiresDateTime: 2022-02-08T22:59:03-00:00
```

```
__*VPN GOING DOWN*__
I’m rebooting the VPN in ten minutes unless anyone objects.
```
4.8. Knock

A knock or ping is a message sent to get the attention of a user or a group of users. It might be sent when a user has not responded to direct messages or mentions, or in a group when something requires the attention of everyone quickly (e.g., a serious unusual situation like a major system outage).

We represent a knock as a text/plain body containing a single CRLF with the alert Content-Disposition token (defined in [RFC3261]).

```plaintext
Content-type: message/cpim
From: <im:alice-smith@example.com>
DateTime: 2022-02-08T22:13:45-00:00
Message-ID: <c1a3375bfe3f@example.com>
Content-Type: text/plain
Content-Disposition: alert
```

4.9. Read Receipt

In instant messaging systems, read receipts typically generate a distinct indicator for each message. In some systems, the number of users in a group who have read the message is subtly displayed and the list of users who read the message is available on further inspection.

Of course, Internet mail has support for read receipts as well, but the existing message disposition notification mechanism defined for email in [RFC8098] is unfortunately inappropriate in this context.

- notifications can be sent by intermediaries
- only one notification can be sent about a single message per recipient
- a human-readable version of the notification is expected
- each notification can refer to only one message
- it is extremely verbose

The proposed format below, message/immi-disposition-notification is sent by one member of an MLS group to the entire group and can refer to multiple messages. There is one IMMI-Disposition line per message, with the disposition of the original message in a parameter. As the disposition at the recipient changes, the disposition can be updated in a subsequent notification.
Content-type: message/cpim
From: <im:bob-jones@example.com>
DateTime: 2022-02-09T07:57:13-00:00
Message-ID: <7e924c2e6ee5@example.com>

Content-Disposition: notification
Content-type: message/immi-disposition-notification
IMMI-Disposition: <4dcab7711a77@example.com>; dispo=read
IMMI-Disposition: <285f75c46430@example.com>; dispo=read
IMMI-Disposition: <c5e0cd6140e6@example.com>; dispo=read
IMMI-Disposition: <5c95a4dfddab@example.com>; dispo=expired

4.10. Attachments

The message/external-body MIME Type is a convenient way to present a URL to download an attachment which should not be rendered inline.

Content-Type: message/external-body; access-type="URL";
URL="https://example.com/storage/bigfile.m4v";
size=708234961

4.11. Conferencing

Joining a conference via URL is also possible. The link could be rendered to the user, requiring a click. Alternatively another Content-Disposition could be specified to more automatic actions. However further calling and conferencing functionality is out-of-scope of this document.

Content-Type: message/external-body; access-type="URL";
URL="https://example.com/join/12345"

5. IMMI CPIM profile

We define a profile of CPIM for instant messaging within MLS. The grammar uses Augmented Backus-Naur Form (BNF) [RFC5234].

5.1. CPIM headers

The following CPIM headers are required:

* From: the identity of message sender. for example im:alice@example.com this identity could be pseudonymous or anonymous if the group policy allows.
* DateTime: the date and time in a reasonable format, as specified in CPIM.
* Message-ID: a message ID which is unique across domains.
* Content-type: As is from CPIM.
* In-Reply-To: Refers to the previous Message-ID. Same semantics as in [RFC5322].
* Supersedes: Refers to the previous Message-ID. Similar semantics to header of the same name in MIXER. Content-Disposition: The intended handling of the message. The two required dispositions are render and reaction.
* Content-Length:

For clarity the grammar for the headers not already included in CPIM are formulated below.

```plaintext
msg-id-header-line = msg-id-header "::" SP msg-id CRLF
msg-id-header = "Message-ID" ; case-sensitive

in-reply-to-header-line = in-reply-to-header "::" SP msg-id CRLF
in-reply-to-header = "In-Reply-To" ; case-sensitive

supersedes-header-line = supersedes-header "::" SP msg-id CRLF
supersedes-header = "Supersedes" ; case-sensitive

msg-id = ""<" id-left "@" id-right ">

id-left = dot-atom-text
id-right = dot-atom-text / no-fold-literal

dot-atom-text = 1*atext *(. "1*atext
atext = ALPHA / DIGIT / atom-symbol

atom-symbol = "!" / "#" / "$" / "%" / "&" / "/" / "*" / ":" / "," / "-' / "'" / "(" / "\" / "\" / "\" / "\" / "\" / "\" / "\" / "\" / "\"

no-fold-literal = "[" "dtext "]"

dtext = %d33-90 / %d94-126 ; Printable US-ASCII
 ; excluding ",", and \\
```

5.2. Definition of message/immi-disposition-notification

The grammar below defines the syntax.
immi-disposition-notification-body = 1*immi-header-line

immi-header-line = immi-header ":" SP msg-id ";" status CRLF

imm-header = "IMMI-Disposition" ; case-sensitive

status = "dispo" "=" status-value

status-value = "read" /  
"error" /  
"delivered" /  
"expired" /  
"deleted" /  
"hidden"

5.3. Required and Recommended MIME types

The following MIME types are REQUIRED:

* message/cpim
* multipart/alternative
* multipart/mixed
* multipart/parallel
* text/plain
* text/markdown

The following MIME types are RECOMMENDED:

* text/html
* message/external-body
* message/immi-disposition-notification
* image/jpeg
* image/png

6. IANA Considerations

6.1. MIME subtype registration of message/immi-disposition-notification

This document proposes registration of a MIME subtype with IANA.

TBC

7. Security Considerations

TBC

8. Normative References
9. Informative References

Mahy
Expires 8 September 2022

[I-D.ietf-mls-protocol]
Barnes, R., Beurdouche, B., Robert, R., Millican, J.,
Omara, E., and K. Cohn-Gordon, "The Messaging Layer
Security (MLS) Protocol", Work in Progress, Internet-
Draft, draft-ietf-mls-protocol-12, 11 October 2021,
<https://datatracker.ietf.org/doc/html/draft-ietf-mls-
protocol-12>.

[I-D.mahy-dispatch-immi-mls-mime]
Mahy, R., "Inside MLS Message Interop (IMMI) MIME type
extensions", Work in Progress, Internet-Draft, draft-mahy-
dispatch-immi-mls-mime-00, 7 March 2022,
<https://datatracker.ietf.org/doc/html/draft-mahy-
dispatch-immi-mls-mime-00>.

Mapping between X.400 and RFC 822/MIME", RFC 2156,
DOI 10.17487/RFC2156, January 1998,

Network Services", BCP 17, RFC 2219, DOI 10.17487/RFC2219,

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,
A., Peterson, J., Sparks, R., Handley, M., and E.
Schooler, "SIP: Session Initiation Protocol", RFC 3261,
DOI 10.17487/RFC3261, June 2002,

Messaging (CPIM): Message Format", RFC 3862,
DOI 10.17487/RFC3862, August 2004,

Specifications: ABNF", STD 68, RFC 5234,
DOI 10.17487/RFC5234, January 2008,

[RFC7763] Leonard, S., "The text/markdown Media Type", RFC 7763,
DOI 10.17487/RFC7763, March 2016,


Author's Address

Rohan Mahy
Wire
Email: rohan.mahy@wire.com
Abstract

This document defines two new extensions to the MLS (Messaging Layer Security) Protocol to allow for negotiation of MIME types exchanged among members of an MLS group.

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 https://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 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
1. 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 RFC 2119 [RFC2219].

The terms MLS client, MLS group, and KeyPackage have the same meanings as in the MLS protocol [I-D.ietf-mls-protocol].

2. Introduction

MLS is a group key establishment protocol motivated by the desire for group chat with efficient end-to-end encryption. While one of the motivations of MLS is interoperable standards-based secure messaging, the MLS protocol does not define or prescribe any format for the encrypted "application messages" encoded by MLS. This document describes two extensions to MLS which allow MLS clients to advertise their supported MIME types, and to specify which MIME types are required for a particular MLS group. These allow clients to discover MLS groups with an interoperable and extensible set of content types.

A companion document [I-D.mahy-dispatch-immi-content] describes a specific profile for interoperable instant messaging body types.

3. Extension Description

This document specifies two MLS extensions of type MimeTypeList: accepted_mime_types, and required_mime_types.

MimeType is the ASCII string encoded as a TLS vector type containing a single MIME type and any of its parameters.

MimeTypeList is an ordered list of MimeType objects.
// Text string representation of a single IANA registered MIME Type.
MimeType mime_type<V> 

struct { 
    MimeType mime_types<V> 
} MimeTypeList 

Example MIME Types:

image/png
text/plain;charset="UTF-8"

An MLS client which implements this specification SHOULD include the accepted_mime_types extensions in its KeyPackages, listing all the MIME types it can receive.

When creating a new MLS group, the group MAY include a required_mime_type extension in the group Extensions. When used in a group, the client MUST include the required_mime_types extension in the list of extensions in RequiredCapabilities.

MLS clients SHOULD NOT add an MLS client to an MLS group with required_mime_types unless the MLS client advertises it can support all of the required MIME Types. As an exception, a client could be preconfigured to know that certain clients support the mandatory types.

4. IANA Considerations

This document proposes registration of two MLS Extension Types.

4.1. accepted_mime_types MLS Extension Type

The accepted_mime_types MLS Extension Type is used inside KeyPackage objects. It contains a MimeTypeList representing all the MIME Types supported by the MLS client publishing the KeyPackage.

Template:
Value: 0x0005
Name: accepted_mime_types

Message(s): This extension may appear in KeyPackage objects
Recommended: Y
Reference: RFC XXXX

Description: list of MIME types supported by the MLS client advertising the KeyPackage
4.2. required_mime_types GroupContext extension

The required_mime_types MLS Extension Type is used inside GroupContext objects. It contains a MimeTypeList representing the MIME Types which are mandatory for all MLS members of the group to support.

Template:
Value: 0x0006
Name: required_mime_types

Message(s): This extension may appear in GroupContext objects
Recommended: Y
Reference: RFC XXXX

Description: list of MIME types which every member of the MLS group is required to support.

5. Security Considerations

The Security Considerations of MLS apply.

Use of the extensions in this document could leak some private information both in KeyPackages and inside an MLS group. They could be used to infer a specific implementation, platform, or even version. Clients should consider carefully the implications in their environment of making a list of acceptable MIME types available.

A client which can take over group administration could prevent members from joining or sending messages in an established group, by requiring a list of required MIME types which the attacker knows is unsupported. This attack is not especially helpful, as taking over group administration can have more disruptive effects.

6. Normative References

[I-D.ietf-mls-protocol]

7. Informative References

[I-D.mahy-dispatch-immi-content]

Author's Address

Rohan Mahy
Wire
Email: rohan.mahy@wire.com
Event Streaming Open Network
draft-spinella-event-streaming-open-network-02

Abstract

This document describes the vision, architecture and network protocol for an Event Streaming Open Network over the Internet.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://example.com/LATEST. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-spinella-event-streaming-open-network/.

Source for this draft and an issue tracker can be found at https://github.com/syndeno/draft-spinella-event-streaming-open-network.

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 https://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 31 July 2022.
Internet-Draft                    ESON                      January 2022

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the
document authors.  All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
license-info) in effect on the date of publication of this document.
Please review these documents carefully, as they describe your rights
and restrictions with respect to this document.  Code Components
extracted from this document must include Revised BSD License text as
described in Section 4.e of the Trust Legal Provisions and are
provided without warranty as described in the Revised BSD License.

Table of Contents

1.  Introduction  . . . . . . . . . . . . . . . . . . . . . . . .   3
2.  An Open Network for Event Streaming over the Internet . . .   4
  2.1.  Free, Open & Neutral Networks (FONN) . . . . . . . . . .   5
  2.2.  Non-discriminatory and open access ..........................   6
  2.3.  Open participation ...........................................   6
  2.4.  Open Access Infrastructure Resources .........................   7
  
  2.4.1.  Open Access DNS Resource Example ..........................   9
  2.4.2.  Flow: Event Streaming Internet Resource .....................   9
3.  Necessities for an Event Streaming Open Network over the
    Internet ............................................................. 11
  3.1.  Necessity 1: Event Streaming Internet Resource Public
        Registry ....................................................... 11
  3.2.  Necessity 2: Establishment of a User Space for Events ... 12
  3.3.  Necessity 3: An Agnostic Subscription Protocol ............. 13
4.  Event Streaming Open Network Architecture ........................ 15
  4.1.  Architecture overview ........................................ 15
  
  4.1.1.  Flow Events Broker (FEB) ................................. 18
  4.1.2.  Flow Name Service (FNS) .................................. 18
  4.1.3.  Flow Namespace Accessing Agent (FNAA) ..................... 22
  4.1.4.  Flow Processor (FP) ...................................... 23
  4.1.5.  Flow Namespace User Agent (FNUA) .......................... 24
  4.2.  Communications Examples ..................................... 24
        4.2.1.  Unidirectional Subscription ........................... 25
        4.2.2.  Bidirectional Subscription ............................ 25
5.  Event Streaming Open Network Protocol ............................. 26
  5.1.  Protocol definition methodology ............................. 26
  5.2.  Flow Namespace Accessing Protocol (FNAP) .................... 27
  5.3.  Implementation ............................................... 28
        5.3.1.  Objectives ............................................. 28
  5.4.  Existing components ......................................... 28
        5.4.1.  Flow Events Broker (FEB) .............................. 29
1. Introduction

Society is rapidly digitalizing and automating the exchanges of value that constitute the economy. Also, considerable time and energy is spent to assure that key transactions can be executed with reduced human involvement with better, faster, and more accurate results. In this context, Event Streaming can play a key role in how the economic system evolves.

However, most of the application layer integrations executed today across organizational boundaries are not in real time. Also, they currently require employing a variety of formats and protocols. Some industries have adopted data formats for exchanging information between organizations, such as Electronic Data Interchange (EDI). However, those integrations are limited to specific use cases and represent a small fraction of all demanded organizational integrations.

Thus, there is no consistent and common consensus on a mechanism for the exchange of events across organizations. This results in a completely custom landscape for each real-time cross-organization integration. In this scenario, development teams must invest plenty of time into understanding and defining a common interface for events exchange.

In this context, we can now introduce how this landscape could change with the introduction of an Event Streaming Open Network over the Internet. When needing to connect real-time event flows across organizations, developers would have a common basis for finding, publishing, and subscribing to event streams. Also, given a set of
standard formats to encode and transmit events, developers could use the programming language of their choice. Overall, this set of standards would drastically reduce the cost of real-time integration, which would also enable experimentation by users.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. An Open Network for Event Streaming over the Internet

In this section, we will argue how Internet standards are developed and why this could be the case for an Event Streaming Open Network.

An interesting example of this phenomenon is the case of ISDN (Integrated Services Digital Network), a set of communications standards for the transmission of voice, video, and data over the PSTN (Public Switched Telephone Network) developed by the ITU-T (Telecommunication Standardization Sector) in 1988. ISDN pretended to use the existing public telephone network to transmit digital data in a time when the Internet connectivity access was not as broadly available as it is today. The main competitor of this standard was the incipient Internet itself, which could be used to transmit the same data.

The Internet alternative needed a protocol to support the same services offered by ISDN, which was initially developed by the conjoint effort of the academic and private sector. Consequently, in 1992 the Mbone (Multicast Bone) was created. This project was an experimental network backbone built over the Internet for carrying multicast IP traffic, which could be used for multimedia content. After some important milestones of this project, the SIP (Session Initiation Protocol) was defined in 1996 and was published as a standard protocol in IETF's [RFC3261]. The reality today is that SIP has completely won the standards battle for multimedia transmission over the Internet, and ISDN usage has been on continuous decline.

As for Event Streaming, we see a similar scenario set-up today. There are currently several open specifications and implementations for Event Streaming, like AMQP (Advanced Messaging Queueing Protocol), supported by RabbitMQ. However, while AMQP can be used for several purposes, Kafka Protocol specializes on Event Streaming Processing and its specialized features make it more convenient than RabbitMQ (i.e. ordering).
In the case of an Event Streaming Open Network over the Internet, if we guide ourselves by the history of the most widely adopted protocols on the Internet, the governance should be similar to that of the WWW or Email. Both the WWW and Email have open specifications as well as open-source implementations. We can mention the Apache Web Server as an open-source implementation of the HTTP protocol; Postfix for SMTP; and Bind for DNS. Nevertheless, the governance for these protocols’ specifications relies on the IETF.

In order to define the characteristics of an Event Streaming Open Network, we will focus on the definition of shared and openly accessible infrastructure. First, we will review the principles of Free, Open & Neutral Networks and why they should be followed for an Event Streaming Open Network. Then, we will show how DNS complies with the criteria to be considered an infrastructure resource. Finally, we will demonstrate how this is also true for Event Streaming.

2.1. Free, Open & Neutral Networks (FONN)

The main principles of a Free, Open & Neutral Network are:

* It is open because it is universally open to the participation of everybody without any kind of exclusion nor discrimination, and because it is always described how it works and its components, enabling everyone to improve it.

* It is free because everybody can use it for whatever purpose and enjoy it independently of his network participation degree.

* it is neutral because the network is independent of the contents, it does not influence them and they can freely circulate; the users can access and produce contents independently to their financial capacity or their social condition. The new contents produced are orientated to stimulate new ones, or for the network administration itself, or simply in exercise of the freedom of adding new contents, but not to replace or to block other ones.

* It is also neutral with regard to the technology, the network can be built with whatever technology chosen by the participants with the only limitations resulting of the technology itself.
2.2. Non-discriminatory and open access

Services such as DNS, the World Wide Web and Email do not discriminate and are open-accessible. Basically, people and organizations can access these networks as long as they can register an Internet Domain and host the required server components. Nowadays, there are alternatives to avoid having to register a domain name to have a web page or an email, such as Cloud WordPress Hosting or Gmail. However, we will focus on the network participants that provide services to end-users.

In the case of Guifi.net, we can highlight how this principle has been adopted in the fact that everybody can take part in the project without discrimination. Moreover, an emphasis is made in easing the participation of the disadvantaged collectives, with less resources or less opportunities to access information technologies, telecommunications, and the Internet.

An Event Streaming Open Network should provide resources in a similar way than the most widely adopted Internet Services. Thus, individuals and organizations must be able to register Flow address spaces for which the existing DNS infrastructure could be leveraged. Moreover, the specification of the protocols that implement the Metadata and Payload formats must also be openly accessible.

2.3. Open participation

Internet Services like DNS, WWW and Email provide individuals and organizations with different ways of participation. First, anybody can obtain the protocols’ specification and build a custom implementation, which would result in a new product compatible with the protocols. Secondly, anybody can register a domain name and set up servers using compatible products. Thirdly, anybody can join and participate in the IETF, the institution that governs the specifications for these protocols.

As for Guifi.net, not only anybody can extend the network with new nodes but also can also participate in existing projects of network extension. Also, the participants can add services on top of the network such as VoIP, FTP servers, broadcast radios, etc.
Regarding active participation on an Event Streaming Open Network, we can highlight the possibility for individuals and organizations to expand the services provided by the open network. This extensibility could be made possible by different uses of the event payloads and will vary significantly depending on the sector. Since we have already proved how Flow is an infrastructure resource, innovation would play its part and its results would be materialized in services expansion.

We can conclude that the same kind of openness of DNS, WWW and Email is necessary for an Event Streaming Open Network. Anybody should be able to obtain the specifications to build an implementation of the service. Also, since it should leverage the DNS infrastructure, anybody would be able to register Flow address spaces. Lastly, the specification could be governed by an institution such as the IETF, due the dependency of Flow with other Internet Services governed by this institution.

2.4. Open Access Infrastructure Resources

The literature about Commons Infrastructure (Frischmann, 2007) defines a set of criteria to evaluate if a resource can be considered an infrastructure resource. This analysis is relevant since it can provide some arguments to prove the need of an infrastructure of commons for Event Streaming, which could then be materialized in an Open Network for Event Streaming. The demand-side criteria for evaluating if a given resource can be considered as an infrastructure resource are:

1. The resource can be consumed nonrivalously.

2. Social demand for the resource is driven primarily by downstream productive activity that requires the resource as an input.

3. The resource is used as an input into a wide range of goods and services, including private goods, public goods and/or non-market goods.

First, a nonrival good describes the "shareable" nature of a given good. Infrastructures are shareable in the sense that the resources can be accessed and used by multiple users at the same time. However, infrastructure resources vary in their capacity to accommodate multiple users, and this variance in the capacity differentiates nonrival resources from partially rival resources. A nonrival resource represents those resources with infinite capacity, while a partially rival resource has finite but renewable capacity. As an example, Broadcast Television is a nonrival resource since additional users do not affect the capacity of the resource. On the
other hand, natural oil resources are completely rival since its availability is limited and it is not renewable. In the middle, we have partially rival resources like a highway, which may be congested. This last characteristic is also true for the Internet since it supports additional users without degrading the service to existing users to a certain extent.

Secondly, infrastructure resources consumption is primarily driven by downstream activities that require this resource as an input. This means that the broad audience consumes infrastructure resources indirectly. For instance, highway infrastructure is used to transport every kind of physical good which people and organizations purchase. This facilitates the generation of positive externalities for society through the downstream production of public goods and non-market goods. These positive externalities might be suppressed under a regime where resource availability is driven solely based on individuals’ willingness to pay.

Regarding willingness to pay, it is relevant to analyze this factor more exhaustively. Frischmann states that if infrastructure access is allocated based on individuals’ willingness to pay the potential positive externalities of that infrastructure might be stifled. Thus, infrastructure resources behave differently than end-user products: if the former are made available solely based on the end-user demands and willingness to pay, those needed infrastructure resources might never be made available. As an example, we can mention that if airports were built based on individuals’ willingness to pay for them, they might not even be built. However, individuals are willing to pay for the airport’s downstream activities, such as purchasing a flight or consuming air-transported goods. Then, a whole set of positive externalities are generated by the existence of an airport in a city.

In the third place, infrastructure resources are used as input for a wide range of outputs. This criterion emphasizes both the variance of the downstream outputs and their nature. Thus, the infrastructure resources possess a high level of genericness which enable productive activities that produce different goods with high variance. If we consider how an airport complies with this criterion, we can mention that not only airports serve individuals that need to travel by air but are also used to transport many kinds of physical goods. These goods then enable other activities throughout the downstream value chain. Then, the output variance of the activities that take airport infrastructure as input is significantly high.
2.4.1. Open Access DNS Resource Example

Now, we will provide as an example how DNS complies with these criteria and why it can be considered an infrastructure resource. 1. DNS infrastructure is a partially rival resource because individuals and organizations can register domains in the Domain Name addressing space. It is partially rival because not every actor can acquire the same domain name. However, the access to registering domain names is open and non-discriminatory. Moreover, DNS is also prone to congestion, which emphasizes its partially rival nature. 2. DNS infrastructure demand is driven principally by downstream products and services. An average Internet user is not paying directly for this infrastructure, but all the Internet services the user consumes pay for DNS infrastructure. This is true for all the Internet services due to the ubiquitous nature of DNS infrastructure. 3. All Internet services take as input DNS infrastructure and produce a broad variety of outputs, which then generate positive externalities to society as a whole by means of private goods, public goods and/or non-market goods.

We can conclude that DNS complies with Frischmann criteria for being considered as an infrastructure resource. The resource is represented both by the domain name that can be and by the querying capacity of DNS servers.

2.4.2. Flow: Event Streaming Internet Resource

In this section, we will describe an Event Streaming Internet Resources. For this, we will consider the previously described guidelines for FONN as well as the characteristics of DNS as a resource. This Event Streaming Internet Resource shall be refered to as "Flow" from now onwards.

To begin with, we need to define what elements could be considered as infrastructure resources in an Event Streaming Open Network. First, the resource must be capable of delivering streams of events to consumers. Secondly, it must also permit producers to write events to the stream. Thirdly, each stream must be identifiable (i.e., URI) and able to be located (i.e., URL). From now on, we will use "Flow" to refer to the infrastructure resource of an Event Streaming Open Network. The first Frischmann criterion requires the resource to be consumed nonrivalously. Complete nonrivalrously for any Internet Service cannot be achieved due to the possibility of congestion and potential unavailability of different elements of the network. The same would be true for a Flow resource. Moreover, the public naming addressing space for Flows would be limited to the same level as that of domain names.
We will continue now with the third criterion. To illustrate the potential of Flow being used as inputs for downstream activities, we will refer to Urquhart’s vision for Event Streaming. He lists two areas in which significant changes can happen:

1. The use of time-critical data for customer experience and efficiency. This is driven because today’s consumers are increasingly expecting great experiences, and organizations are almost always motivated to improve the efficiency of their operations.

2. The emergence of new businesses and business models. Businesses and institutions will quickly discover use cases where data processed in a timely manner will change the economics of a process or transaction. They may even experiment with new processes, made possible by this timely data flow. Thus, flow resources will also enable innovation. These innovations are responsible for generating positive externalities.

Then, we have demonstrated why Flow resources can be considered as infrastructure resources using Frischmann’s Demand-side Theory of Infrastructure. These resources can be managed in an open manner to maximize positive externalities, which basically means maintaining its open access, not discriminating, and eliminating the need to obtain licenses to use the resources. Consequently, managing infrastructure resources in this manner eliminates the need to rely on either market actors or governments.

Lastly, the adoption of an Event Streaming Open Network implies taking Flow resources as inputs for productive activities. These inputs would then be used downstream to generate private goods, public goods and/or non-market goods. Additionally, we can assure that most of the consumers of Flow would not directly consume Flow resources. They would consume the outputs of downstream activities that use Flow as input. Again, the consumers may not be willing to pay for Flow resources directly.

We can conclude this section mentioning that an Event Streaming Open Network would enable one infrastructure resource called Flow. The access to this resource can be managed in an openly manner: maintaining open access, not discriminating users or different uses of the resource, and eliminating the need to obtain approval or a license to use the resource.
3. Necessities for an Event Streaming Open Network over the Internet

In this section, we will describe the main needs for the broad adoption of Event Streaming. The focus will be made on detecting and describing the missing capabilities that could not only enable but also accelerate the event data integration among different organizations. The different necessities detailed in this section will serve as input for an architecture design.

3.1. Necessity 1: Event Streaming Internet Resource Public Registry

A public registry of an organization’s available event streams does not exist. We will argue in this section why this is the core component that an Event Streaming Open Network can provide.

Nowadays, when an organization needs to publish an event stream or event flow, they usually follow some form of the following steps:

1. Develop and deploy a producer application that writes events to a queue.

2. Create all necessary networking permissions for external public access to the queue.

3. Inform the remote user the access information (i.e., Hostname/IP, protocol, and port) together with the required client details and technology for accessing the stream (i.e., Apache Kafka Protocol, RabbitMQ API, etc.).

4. Create credentials for consumer authentication and authorization access to the queue. 5. Develop and deploy a consumer application that reads the queue.

Now, we can compare this process to a simple email interaction: 1. Sender opens a graphical Mail User Agent application and sends an email to an email address formatted as user@domain. 2. The message is sent to an SMTP server that routes it to the destination SMTP servers for the given domain. Once received, the message is put into the user mailbox. 3. When the recipient checks its mailbox by IMAP or POP3, the new email is transferred to the Mail User Agent.

In these two scenarios, we can see that the information needed to be exchanged offline by the actors is completely different in size and content.

First, in the case of email, there is a shared naming space given by the Domain Name Service (DNS). The email format has been standardized by the IETF in [RFC5321], section 2.3.11. Thus, there
is a common naming space that is used for referencing mailboxes in the format user@domain. Thus, the offline details communicated by the peers is only the recipient email address. There is no analogous standard nor an open alternative for Event Streaming.

Therefore, in the case of Event Streaming, users need to perform plenty of offline communication to agree not only on the technology to use but also on the queue to use. For instance, two organizations may be currently using Apache Kafka and need to share an event stream among themselves. The organization having the source of the stream should provide the following details to the consumer organization:

- **Topic or Queue name**: name of the topic resource in the Apache Kafka Cluster
- **Authentication information**: User and password, TLS Certificate, etc.

In the case these organizations were not both using Apache Kafka, the use case cannot be simply solved without incurring in development or complex configurations as well as adopting proprietary components.

We can conclude that an Event Streaming Open Network should provide a global accessible URI for streams in a similar fashion than email, to reduce offline developers’ interactions. This means being able to name event streams in a common naming space like DNS, as well as providing a mechanism for users to discover the location and connections requirements.

### 3.2. Necessity 2: Establishment of a User Space for Events

Another need for broad adoption is due to the inexistence of a common and agreed user convention. In the general literature, we cannot find reference to the types of users that would consume or produce events to and from an event stream.

In this sense, it is also appropriate to consider the email use case. Basically, an email user only needs to know the email address, the password, the URL of a web mail client or the details of IMAP/POP3 server connection. Once the user has this information, it’s possible to access an email space or mailbox where the user can navigate the emails in it. Also, IMAP provides the possibility for the user to create folders and optionally share them with other users.

There is no analogous service currently available for Event Streaming analogous to the email case. This means that the user concept in Event Streaming is limited to authentication and authorization.
Thus, the user does not have access to a "streambox". The result is the impossibility for a person or an application to possess a home directory containing all the streams owned by the user.

As a conclusion for this section, we can mention that it is necessary to embrace a user space resource for Event Streaming. This resource should not only solve the users' motivations and requirements but also reduce the offline verbal communications and custom development dependencies. In the next sections, we will refer to this component as the Event User Space Service.

3.3. Necessity 3: An Agnostic Subscription Protocol

A third need for wide adoption is an agnostic protocol to manage subscriptions to event streams. For this need to be solved, it would be necessary first to count with an Event User Space Service. Then, in case a user has created a stream and wants to enable public subscriptions by other users, there is no general protocol to inform other parties of this subscription intention nor its confirmation.

The result is the inability for the users to seamlessly subscribe to an event stream. They either must employ protocols like MQTT or, in the need of employing other application protocols like Apache Kafka, hardcode the subscription details in the different software implementations. This means that there is no general subscription protocol for Event Streaming that is agnostic of the application protocol employed. This protocol implements both the Metadata Payload Format and Payload Format.

A good example to illustrate the difference between a control protocol that implements a Metadata Payload Format from a payload protocol that implements a Payload Format is how SIP (Session Initiation Protocol) works with RTP (Real Time Protocol) to provide VoIP capabilities. The former is a control protocol that initiates and maintains a session or call while the latter is the one responsible for carrying the payloads, which in the case of VoIP it would be coded audio.

Consequently, a similar definition of protocols could potentially mitigate this limitation for Event Streaming. If a protocol can be used to establish and maintain the subscriptions relationships while another different protocol is used for the events payload, all the current application protocols implementations could be supported.
Additionally, by counting also with an Event Streaming Public Registry, it would be possible to provide URI for streams in a similar way as email works with the "mailto" URI. For instance, in web pages one can find that email addresses are linked to mailto URIs which, when clicked, open the default email user application (i.e., Microsoft Outlook) to send an email to the referenced email address.

If a user counts with a user space or streambox, then a user application like an email client could provide access to it. Then, if the user clicks on a link of a stream URI (i.e. "stream:myeventflow"), the streambox application would open and subscribe to the given stream.

Currently, the Metadata Payload Format as well as the Payload Format are both provided by the queue or log application protocol. In the case of Apache Kafka, both formats are implemented within the Apache Kafka Protocol. This introduces a barrier for interoperability among different technologies, meaning that flows of event data cannot be seamlessly connected, without relying on custom development or proprietary software licensing.

We can conclude that there is an actual need for an open specification of an Event Subscription Service for event streams, which implements what Urquhart calls Metadata Payload Format. This specification could be materialized in a network protocol that introduces an abstraction for the event queue or log technologies implemented by different organizations.

3.4. Necessity 4: An Open Cross-sector Payload Format

Currently, the different implementations of Event Streaming combine both the Payload Format with the Metadata Format. This means that the same protocol utilized for payload transport is used for subscription management.

When a producer intends to publish events to a queue or, using Apache Kafka terminology, when a producer intends to write records to a topic, first it needs to initiate a connection to at least one of the Apache Kafka Brokers. In that initial exchange of TCP packages, the producer is authenticated, authorized, and informed with topic details. This set of transactions would belong to a protocol that implements a Metadata Payload Format. Afterwards, when the Producer starts writing the events to the topic, it encapsulates the event payload in a Kafka Protocol message. This latter behavior makes use of a Payload Format. Thus, we can observe how both theoretical formats are coupled in a single protocol. Similar behavior of a coupled Metadata and Payload Format in one single protocol happens also in AMQP, MQTT and RabbitMQ.
As for the consumer, the behavior is the same with the difference that
the initial intention is to subscribe to a queue or, in Apache
Kafka terminology, to consume records of a topic. Then, a set of TCP
packages encapsulating the Apache Kafka protocol authenticates,
authorizes, and informs the Consumer with topic details for
consumption. Afterwards, the consumer can start polling for new
records in the different partitions of the topic. It is worth
mentioning that the consumer needs to implement more queue management
logic than the Producer, especially when multiple replicas of a
consumer type are deployed.

If we focus on the Payload Format, there is the need for an
implementation-agnostic payload format suitable for Event Streaming.
In this sense, CloudEvents project of the CNCF proposes a
specification and a set of libraries for this purpose. The goal is
to use CloudEvents specification as a Payload Format regardless of
the Payload Protocol being used. For instance, we could transmit
events in the CloudEvents format using the Kafka or AMQP Protocol.

The general structure of the CloudEvents Payload Format includes a
standardized methodology to include event data in an event message.
For instance, instead of defining a customized JSON structure for
sending the events of temperature changes measured by a device, a
CloudEvent object could be used. Temperature could be included as an
attribute in the CloudEvent object.

We can then conclude that while there is no current protocol
candidate that implements the Metadata Format, CloudEvents is a good
candidate for the Payload Format needed in an Event Streaming Open
Network. In this way, the different CloudEvents libraries made
available in several programming could be leveraged.

4.  Event Streaming Open Network Architecture

In this section, we will describe the overall architectural proposal
for an Event Streaming Open Network. This description will include
the different actors in play, the software components required, as
well as the network protocols that should be specificized.

4.1.  Architecture overview

In Figure 1 we illustrate a high-level overview of an architecture
proposal for the Open Network.

(Artwork only available as svg: No external link available, see
draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 1: Figure 1
We can identify different Network Participant (NP) in Figure 1 represented by different colors. The different NPs act as equals when consuming or producing events as part of the Flows they own. All of NPs implement the Event Streaming Open Network Protocol, which is described in the next chapter.

In the diagram, an initial flow starts on the orange NP to which a user in the blue NP is subscribed. After processing the events received in the first flow, the results are published to a new flow in NP blue, to which the orange NP is subscribed as well. Now, the green participant is subscribed to the same flow, enabling downstream activities across the rest of the network participants.

It is possible to observe how the high-level architecture allows sharing the streaming of events across different network participants and their users. Also, there is also the need for security, in order to allow or deny the access to write to and read from flows.

Regarding security, the architecture considers the integration with an Identity & Access Management service, which could implement popular protocols such as OAuth, SAML or SASL. However, the network should also enable anonymous access in the same way FTP does. This means that a given NP could publicly publish flow and allow any party to subscribe to it.

For example, nowadays the Network Time Protocol (NTP) is used to synchronize the day and time on servers. There are many NTP servers available that allow anonymous access, meaning that the service is openly available. The same must be considered for the Event Streaming Open Network.

Additionally, the NP must be able to expand the capacity to support any number of flows, as well as extending the network with new services. Not only NP must be able to include any given set of data within events but also, they must be able to build applications and services on top of the network by employing the architecture primitives.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 2: Figure 2

Now, we provide a brief description of all the components that appear in the diagram of Figure 2. In the next sections further details of the components are provided.
* Flow Events Broker (FEB): a high-available and fault-tolerant service that provide queues to be consumed by network services, by users, and their applications. An example of an Event Queue Broker can be Apache Kafka, AWS SQS or Google Cloud PubSub. The payload format implemented by these tools are what in 3.1.4 we called Event Streaming Payload Format.

* Flow Name Service (FNS): a DNS-based registry that acts as an authoritative server for a set of domain names, which are used to represent flow addresses in a flow namespace. These domains contain all the necessary information to resolve flow names into flow network locations. This component refers to what in 3.1.1 we named Event Streaming Registry.

* Flow Namespace User Agent (FNUA): an application similar to User Mail Agents like Microsoft Outlook or Gmail. This application provides access to flow namespaces to users of the network. The definition of this component implies the specification of a dedicated protocol. We will refer to this protocol as FNAP (Flow Namespace Accessing Protocol).

* Flow Namespace Accessing Agent (FNAA): the server-side of the Flow Namespace User Agent. This component is the one that must provide convenient integration methods for GUI. This component refers to what in 3.1.2 we named Event User Space Service. This component must implement the same protocol selected for the Flow Namespace User Agent: FNAP (Flow Namespace Accessing Protocol).

* Flow Processor (FP): a flow processing instance used to set up subscriptions that connect local or remote flows on demand. This component implements the processing part of what in 3.1.3 we called Event Subscription Service. This component will be created and managed by a FNAA instance, and the communication is held through an Inter-process Communications (IPC) interface. Also, this service must implement an Event Payload Format, for which we will mainly consider CNCF’s CloudEvents and Protobuf.

* Flow Namespace Accessing Protocol (FNAP): the protocol implemented in the Flow Namespace Accessing Agent as well as in the Flow Namespace User Agent. The former will act both as a server and a client while the latter only as a client. This protocol is described in the next chapter.
4.1.1. Flow Events Broker (FEB)

The FEB implementation that we will mostly consider is Apache Kafka. This open-source project is quickly becoming a commodity platform, and major cloud providers are building utilities for it. However, as a design decision, it should be possible to use the same protocols to support other applications, such as RabbitMQ, Apache Pulsar or the cloud-based options like AWS SQS or Azure Events Hub.

Apache Kafka is the ecosystem leader in the Event Streaming space, considering mainly adoption. There is a growing set of tools and vendors supporting its installation, operation, and consumption. This fact makes Apache Kafka much more appealing to enterprise developers. However, the broker should provide a common set of functionalities which can be seen in the diagram of Figure 3.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 3: Figure 3

The selection of the Events Broker will impact on the implementation of the Flow Namespace Accessing Agent. This last component will be responsible for knowing how to set up and manage flows on top of different Events Brokers.

4.1.2. Flow Name Service (FNS)

FNS is a core component for the overall proposed architecture. This component provides all needed functionalities for obtaining Flow connection details based on a Flow URI (Uniform Resource Identifier). Thus, it is required to define a URI format for Flow resources and to specify mechanisms for resource location resolution.

In this section, we will focus on describing both the URI for Flow as well as the DNS mechanism for obtaining Flow network location details.

4.1.2.1. Leveraging DNS infrastructure

As mentioned previously, this component must maximize its leverage on the existing Internet DNS infrastructure. The reason for this requirement is to avoid defining new protocols and services that prevent broad adoption. Currently, DNS is the de facto name resolution protocol for the Internet, and there exist libraries for its usage on every programming language.
Whereas DNS is mainly used to resolve FQDN (Fully Qualified Domain Names) into IP addresses, there are many other functionalities provided by the global DNS infrastructure. Theoretically, DNS is an open network of a distributed database. Individuals and organizations that want to participate in the network need to register a domain name and set up Authoritative DNS servers for domains.

It is not in the scope of this work to detail the different available usages of DNS functionalities, but we can mention that it provides special Resource Records (i.e., types of information for a FQDN) that are solely used by special protocols. For instance, the MX Resource Records are used by SMTP servers to exchange email messages.

For the Flow Open Network, it will be required to define a URI format for flows as well as the mechanism to resolve an URI into all the needed information to connect to a flow. In the case of email, a URI is the email address while the connection details will be the SMTP server responsible for receiving emails for that account. For instance, an email URI could be user@domain.com while its connection details could be smtp://mail.domain.com. The way in which the connection details are obtained is by resolving the MX DNS Resource Records of domain.com, which in this example is mail.domain.com.

4.1.2.2. Flow URI

As we mentioned previously, the first needed element is a URI definition for flow resources. These resources identification must capture the following details: * Domain, a registered domain in which create flow resources references. For example, airport.com. * Flow Namespace, a subdomain which is solely used by users to host flow names. This subdomain must be delegated to the Flow Name Server component and desirable should not be used for any other purpose other than flow. * Flow Name, a name for each flow that must be unique within its domain. The combination of flow name and flow domain results in an FQDN. For instance, we could have a flow named arrivals of the domain flow.airport.com. Thus, the FQDN of the flow would be arrivals.flow.airport.com. Also, the name can contain dots so that the following FQDN could be also used: airline.arrivals.flow.airport.com.

Thus, the general syntax of a flow URI would be:

```
flow://flow_name.flow_namespace.domain
```

This URI has the advantage that is similar to "mailto" URI and could be implemented in HTML to refer to flow resources. Some examples:
The flow URI must unequivocally identify a flow resource and provide, by means of DNS resolution mechanisms, all the information required to use the flow. Among these parameters, at least the following should be resolvable:

* Event Queue Broker protocol utilized by the flow. For instance, if Apache Kafka is used, the protocol would be "kafka"; In case RabbitMQ is used by the flow, "amqp". Also, it must be informed if the protocol is protected by TLS.

* Event Queue Broker FQDN or list of FQDNs that resolve to the IP address of one or a set of the Event Queue Brokers. For instance, kafka-1.mycompany.com, kafka-2.mycompany.com.

* Event Queue Broker Port used by the Event Queue Brokers. For instance, in the case of Kafka: 9092, 9093.

* Event Queue Broker Transport Security Layer can be implemented. Thus, it is needed to know if the connection uses TLS before establishing it.

* Queue Name hosted in the Event Queue Broker, which must be equal to that of the corresponding flow name.

The general syntax of the Flow URI would be as follows:

`flow://flowName.flowCategory.myNameSpace.domain.tld`

* Flow Namespace FQDN: myNameSpace.domain.tld

* Flow Name: flowName.flowCategory

* Flow FQDN: flowName.flowCategory.myNameSpace.domain.tld

The following are examples of this URI Syntax:

`flow://notifications.calendar.people.syndeno.com`

* Flow Namespace FQDN: people.syndeno.com
4.1.2.3. Flow name resolution

In Figure 4, we can see how a Flow FQDN can be resolved by means of the Flow Name Service.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 4: Figure 4

In order to illustrate the Flow Name resolution procedure by the FNAA (Flow Namespace Accessing Agent), we can consider the following flow URI:

flow://notifications.calendar.people.syndeno.com

First, the FNAA will perform a query to the DNS resolvers. These will perform a recursive DNS query to obtain the authoritative name servers for the Flow Namespace: people.syndeno.com. Thus, the authoritative name servers for syndeno.com will reply with one or more NS Resource Record containing the FQDN for the authoritative name servers of people.syndeno.com.

Secondly, once these name servers are obtained, the FNUA will perform a PTR query on the Flow FQDN adding a service discovery prefix. The response of the PTR query will return another FQDN compliant with SRV DNS Resource Records [RFC2782] and DNS Service Discovery [RFC6763].

In this case, the query for PTR records would be as follows: """" ;; QUESTION SECTION: ;notifications.calendar.people.syndeno.com. IN PTR """""""" The response would be in the following form: """""""" ;; ANSWER SECTION: notifications.calendar.people.syndeno.com. 21600 IN PTR _flow._tcp.notifications.calendar.people.syndeno.com. """""""" Using the FQDN returned by this query, an additional query asking for SRV records is made: """""""" ;; QUESTION SECTION: ;_flow._tcp.notifications.calendar.people.syndeno.com. IN SRV
First, the response informs the network location of the FNAA server, in this case a connection should be opened to TCP port 65432 of the IP resulting of resolving fnaa.syndeno.com:  

Secondly, this response offers other relevant information, like the TCP port where the queue service is located (9092). It also includes a TXT Resource Record that establishes the protocol of the Event Queue Broker, defined in the variable "broker-type=kafka".

Now, using the returned FQDN for the queue, kafka.syndeno.com, the resolver can perform an additional query:  

The Flow Namespace Accessing Agent is the core component of a Network Participant. This server application implements the Flow Namespace Accessing Protocol that allows client connections.

In the diagram of Figure 5 we can see the different methods that the FNAA must support.

The clients connecting to a FNAA server can be remote FNAA servers as well as FNUA. The rationale is that users of a NP connect to the FNAA by means of a FNUA. On the other hand, when a user triggers a new subscription creation, the FNAA of his NP must connect as client to a remote FNAA server.
4.1.4. Flow Processor (FP)

Whenever a new subscription creation is triggered and all remote flow connection details are obtained, the FNAA needs to set up a Processor for it. The communications of the FNAA to and from the FP is by means of an IPC interface. This means that there can be different implementations of Processors, one of which will be the Subscription Processor.

In the diagram of Figure 6, we can see the initial interface methods that should be implemented in a Flow Processor.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 6: Figure 6

Depending on the use of the processor, different data structures should be added to the different methods. In the case of a Subscription Processor, the minimum information will be the remote and local Flow connection details. Moreover, the interface also should include methods to update the Processor configuration and to destroy it, once a subscription is revoked. Finally, due to the nature of the stream communication, there could also be methods available to pause and to resume a Processor.

There can be different types of Processors, which we can see in Figure 7.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 7: Figure 7

In Figure 7, we can see that there are different types of Flow Processors: * Bridge Processor: Consumes events from a Flow located in an Event Broker (i.e., Apache Kafka) and transcribes them to a single Flow (local or remote). * Collector Processor: Consumes events from N Flows located in an Event Broker and transcribes the aggregate to a single Flow (local or remote). * Distributor Processor: Consumes events from a single Flow and transcribes or broadcast to N Flows (local or remote). * Signal Processor: Consumes events from N Flows and produces new events to N Flows (local or remote)

To implement the previously described Subscription Processor, we can utilize some form of the Bridge Processor. Although we are initially considering the basic use case of subscription, it must be possible for the network to extend the processor types supported. In any
case, the different FNAA servers involved must be aware the supported processor types, with the goal of informing the users the capabilities available in the FNAA server. For instance, the fact that a FNAA supports the Bridge Processor should enable the subscription commands in the FNAA, for users to create subscriptions using the Bridge Processor.

In summary, the IPC interface should support all the possible processors that the network may need although we are initially considering the subscription use case.

4.1.5. Flow Namespace User Agent (FNUA)

The FNUA is an application analogous to email clients such as Microsoft Office or Gmail. These applications implement either different network protocols to access mailboxes by means of IMAP and/or POP3. In the case of FNUA, the protocol implemented is the FNAP (Flow Namespace Accessing Protocol).

The FNUA is an application that acts as a client for the FNAA server. Only users that possess accounts in a Network Participant should be able to login to FNAA to manage Flow Namespaces. The FNUA could be any kind of user application: web application, desktop application, mobile application or even a cli tool.

In the Diagram of Figure 8 we can see the actions that the user can request to the FNUA.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 8: Figure 8

The main goal of the FNUA is to provide the user with access to Flow Namespaces and the flows hosted in them. A user may have many Flow Namespace and many Flows in each of them. By means of the FNUA, the user can manage the Flow Namespaces and the Flows in them. Also, the FNUA will provide the capabilities required to subscribe to external Flows, whether local to the FNAA, local to the NP or remote (in a different NP FNAA server).

4.2. Communications Examples

In this section, two usage examples of Network Participants communications are provided. The first one, we call unidirectional, since one NP subscribes to a remote Flow of a different NP. The second one, we call it bidirectional, since now these NP have mutual subscriptions.
4.2.1. Unidirectional Subscription

In the diagram of Figure 9, we can see an integration between two NP. In this case, there is a FlowA hosted in the Orange NP to which the FlowB in the Blue NP is subscribed. Both FlowA and FlowB count with a queue hosted in the Flow Events Broker, which could be an Apache Kafka instance for example. However, it must be possible to employ any Flow Events Broker of the NP’s choice.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 9: Figure 9

The steps followed to set up a subscription to a remote flow are: 1. A user of the Blue NP creates a new subscription to remote FlowA by means of the Flow Namespace User Agent (FNUA). 2. The FNUA connects to the Flow Namespace Accessing Agent (FNAA) of the Blue NP to inform the user request. 3. The FNAA in the Blue NP discovers the remote FNAA to which it must connect to obtain the flow connection parameters. First, it needs to authenticate and, if allowed, the connection parameters will be returned. 4. Once the FNAA in the Blue NP has all the necessary information, it will set up a new Processor that connects the flow in the Orange NP to a flow in the Blue NP. 5. Once the subscription is brought up, every time a Producer in the Orange NP writes an event to FlowA, the Flow Processor will receive it, since it is subscribed to it. Then, the Flow Processor will write that event to FlowB in the Blue NP. 6. From now on, every Consumer connected to FlowB will receive the events published on FlowA.

In case the user owner of FlowA in the Orange NP wishes to revoke the access, it must be able to do so by means of security credentials revoking against the Identity & Access Manager of the Orange NP.

4.2.2. Bidirectional Subscription

In Figure 10 we can see an example of all the components needed to set up a flow integration between two different NP. In this case, there are two flows being connected: * FlowA of the Orange NP with FlowB of the Blue NP * FlowC of the Blue NP with FlowD of the Orange NP

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 10: Figure 10
Each Flow has its corresponding Queue hosted in the NP Flow Events Broker. Also, there is one Flow Processor for each connection, meaning that these components are in charge of reading new events on source flows to write them to the destination flows as soon as received.

Also, we can see that in order to connect FlowB to FlowA, a connection from the Blue NP’s FNAA has been initiated against the Orange NP’s FNAA. This connection uses the FNAP to interchange the flow connection details. Analogously, the FNAA connection to set up the integration of FlowC with FlowD has been initiated by the Orange NP’s FNAA.

After the flow connection details are obtained, the different Flow Processors are set up to consume and produce events from and to the corresponding Queue in the different NPs.

Once the two processors are initialized, all the events produced to FlowA in the Orange NP will be forwarded to FlowB in the Blue NP; and all the events produced to FlowC in the Blue NP will be forwarder to FlowD in the Orange NP.

5. Event Streaming Open Network Protocol

The protocol to be used in an Event Streaming Open Network is a key component of the overall architecture and design. This chapter is dedicated to thoroughly describe this protocol.

5.1. Protocol definition methodology

It is now necessary to specify the protocol needed for the Flow Namespace Accessing Agent or FNAA, which we have named the Flow Namespace Accessing Protocol or FNAP. In the diagram of Figure 11 we can see how an FNAA client connects with a FNAA server by means of the FNAP.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 11: Figure 11

In order to define a finite state machine for the protocol and the different stimuli that cause a change of state, the model presented by M.Wild (Wild, 2013) in her paper "Guided Merging of Sequence Diagrams" will be employed. This model is beneficial since it provides an integrated method both for client and server maintaining the stimuli relationship that trigger a change of state in each component.
In Figure 12 we have the method proposed by Wild for SMTP, in which there are boxes representing states and arrows representing transitions. Each transition has a label composed of the originating stimulus that triggers the transition and a subsequent stimulus effect triggered by the transition itself. For instance, when a client connects to an SMTP Server, the client goes from "idle" state to "conPend" state. The label of this transition includes "uCon" as the stimulus triggering the transition, which triggers the effect "sCon". Then, on the diagram for the server we can see that the "sCon" triggers the transition from "waiting" state to "accepting" state in the server.

This method will be used to define the states and transitions for the Flow Namespace Accessing Protocol both for client and server.

5.2. Flow Namespace Accessing Protocol (FNAP)

Using the model proposed by Wild described previously, we define the finite-state machine for the FNAA Server, which we can see in Figure 13.

The model in right side of Figure 13 shows that the FNAA server starts in a "waiting" state, which basically means that the server has successfully set up the networking requirements to accept client connections. Then, when a client connects, a transition is made to "accepting" state, in which internally the authentication procedure is made. If the authentication is successful, a transition is made to "ready" state, meaning that the client can now execute commands on the FNAA server.

For each command that the client executes, a transition is made to "cmdRecvd" state. Then, a response is returned to the client, transitioning again to "waiting" state. When the client executes the "Quit" command, a transition is made to the "waiting" state and the server must free all used networking resources for the now closed connection.
On the left side of Figure 13, we also have the client state machine with its corresponding transitions. The client triggers a connection to the server and once established, an authentication is needed. Once the authentication is correctly done, the client can start requesting commands to the server. For each command executed by the client, a transition is made to "cmdPend" state, until a response is returned by the server.

Eventually, a "Quit" command will be executed by the client and the connection will be closed.

5.3. Implementation

In this section, we provide an approach for the overall implementation of the proposed Event Streaming Open Network. Considering the components defined previously for the architecture, we will define which existing tools can be leveraged and those that require development.

5.3.1. Objectives

The objective of this implementation is to provide specifications for an initial implementation of the overall architecture for the Event Streaming Open Network. Whenever it is possible, existing tools should be leveraged. For those components that need development, a thorough specification is to be provided.

5.3.1.1. Implementation overview

In Figure 14, we have a diagram of the overall implementation proposal. The components that have the Kubernetes Deployment icon are the ones to be managed by the FNA server instance. Then, we have a Kafka Cluster that provides a Topic instance for each flow. Finally, the DNS Infrastructure is leveraged.

(Artwork only available as svg: No external link available, see draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 14: Figure 14

5.4. Existing components

In this section, we describe the existing software components that can be leveraged for implementation.
5.4.1. Flow Events Broker (FEB)

Since there are currently many implementations for this component, it is necessary to develop the needed integrations of other components of the architecture to the main market leaders. Thus, we will consider the following Flow Events Broker for the implementation: Apache Kafka, AWS SQS and Google Compute PubSub.

In summary, this component does not need to be developed from scratch. However, the FNAA will need to be able to communicate with the different Flow Events Broker, meaning that it must implement their APIs as a client.

5.4.2. Flow Name Service (FN)

This component can be completely implemented by leveraging on the ISC Bind9 software component, which is the de facto leader for DNS servers. A given NP will need to deploy a Bind9 Nameserver and enable both DNSSEC and DNS Dynamic Update.

The impact of adopting Bind9 for the implementation means that the FNAA component needs to be able to use a remote DNS Server to manage the Flow URI registration, deregistration and execute recursive DNS resolution.

5.4.3. Components to be developed

In this section, we describe a set of tools that require development. These components, especially the FNAA, are the core components of every Network Participant. Moreover, these are the components that implement the network protocol FNAP.

Since these are the core components of the network, they are the natural candidates for validation. In the next chapter, we will show the feasibility of the core network components in the form of a Proof of Concept.

5.4.3.1. Flow Namespace Accessing Agent (FNAA)

The Flow Namespace Accessing Agent is a server component that triggers the creation of child processes that implement the different Flow Processors. This means that the instance running the FNAA will bring up new processes for each processor. One way of implementing this functionality can be a parent process that creates new child processes for each processor. However, this would imply the need of creating and managing different threads in a single FNAA instance.
The problem with the approach of a parent process and child processes for the FNAA is on the infrastructure level. The more processor a FNAA needs to manage, the more compute resources the FNAA would need. In the current cloud infrastructure context, this is a problem because it means that additional compute resources should be assigned to the FNAA, depending on the quantity of processors and the required resources for each of them. In summary, this approach would be vertically scalable but not horizontally scalable.

Then, to avoid the scalability issue, the approach we propose is by implementing a Cloud Native application. By leveraging on Kubernetes, it is possible to trigger the creation of Deployments, which are composed of Pods. Each Pod can contain a given quantity of containers, which are processes running in a GNU/Linux Operating System. In this way, we can dedicate a Pod to run the FNAA server and different Pods to run the Processors. This approach provides a convenient process isolation and enables both horizontal and vertical scalability.

Moreover, the way in which the FNAA would bring up and manage Processor instances would be through an integration with the underlying Kubernetes instance, by means of the Kubernetes API. The result is a Cloud Native application that leverages the power and flexibility of Kubernetes to manage the Processor instances.

On the other hand, the programming language for the FNAA must also be defined. For this, we consider that it must be possible to implement the FNAA and the Flow Processors in different programming languages. For the FNAP it is recommended to employ Golang, since Kubernetes CLI tool is implemented in this language and there are several libraries available for integration. As for the Flow Processors, it must be possible to use any programming language as long as the IPC interface is correctly implemented.

Regarding the IPC interface for the communications between the FNAA and the Flow Processors, the recommendation is to employ gRPC together with Protobuf. The rationale for choosing this technology is the fact that gRPC enables binary communications, which are the desired type of communication for systems integration. Then, both the FNAA and the Flow Processors must share this Protobuf interface definition and implement it accordingly through gRPC.

Finally, the FNAA must implement the protocol we have named FNAP, which provides the main set of functionalities for the Event Streaming Open Network. The implementation of FNAP must be stateful, in the sense that it is connection-based. Additionally, the implementation must be text-based, with the goal that humans can interact with FNAA servers in the same way that it is possible for
SMTP servers. The transport protocol must be TCP with no special
definition for a port number, since the port should be able to be
discovered by means of DNS SRV Resource Records.

Regarding security for the FNAA servers, TLS must be supported. This
means that any client can start a TLS handshake with the FNAA servers
before issuing any command.

In conclusion, the implementation of the FNAA over Kubernetes
provides the needed flexibility and set of capabilities required for
this component. It is recommended to implement the FNAA in Golang
and enable the implementation of Flow Processors in any programming
language as long as the Protobuf interface is correctly implemented.
Finally, the FNAA must implement the protocol FNAP in a connection-
based and text-based manner.

5.4.3.2. Flow Namespace User Agent (FNUA)

The Flow Namespace User Agent (FNUA) can have different
implementations as long as they comply with the protocol FNAP.

We propose the initial availability of a CLI tool that acts as a Flow
Namespace User Agent. This CLI tool must provide a client
implementation of all the functionalities available in the FNAA
server. Among the functionalities to be implemented as a must, we
can mention: * Discover the FNAA server for a given Flow URI. *
Connect to the FNAA server to manage Flow Namespaces and Flows, as
exemplified in Figure 8.

Additionally, the FNUA should be able to discover the Authoritative
FNAA server for a given Flow Namespace. This discovery shall be
performed by leveraging on the DNS-SD specification. Refer to Annex
D to review the discovery process.

Regarding the implementation of the CLI tool, it is recommended to
employ Golang together with Cobra, a library specialized to create
CLI tools. In Figure 15 we have a diagram that shows the different
functionalities that the CLI tool should implement.

(Artwork only available as svg: No external link available, see
draft-spinella-event-streaming-open-network-02.html for artwork.)

Figure 15: Figure 15
6. Proof of Concept

In this section, we will focus on providing a minimum implementation of the main Event Streaming Open Network component: the Flow Namespace Accessing Agent. This implementation should serve as a Proof of Concept of the overall Event Streaming Open Network proposal.

As described in the previous section, the Flow Namespace Accessing Agent (FNAA) is the main and core required component for the Open Network. All Network Participants must deploy an FNAA server instance in order to be part of the network. The FNAA actually implements a server-like application for the Flow Namespace Accessing Protocol (FNAP). Then, the first objective of this Proof of Concept is to show an initial implementation of the FNAA server component.

On the other hand, the FNAA is accessed by means of a Flow Namespace User Agent (FUA). This component acts as a client application that connects to a FNAA. Also, this component can take different forms: it could be a web-based application, a desktop application or even a command line tool. For the purposes of this Proof of Concept, we will implement a CLI tool that acts as a client application for the FNAA. Thus, the second objective of this PoC is to provide an initial implementation of the FNUA client component.

In the following sections, we will first describe the minimum functionalities considered for validating the overall proposal for the Event Streaming Open Network. This minimum set of requirements for both the FNAA and the FNUA will compose the Proof of Concept.

Afterwards, we will describe the technology chosen for the initial implementation of both the FNAA and the FNUA. Then, a description of how these tools work in isolation will be provided. Subsequently, we will review different use cases to prove how the network could be used by network participants and its users.

Lastly, we will provide a conclusion for this Proof of Concept, where we mentioning if and how the minimum established requirements have been met or not.

6.1. Minimum functionalities

Network Participants system administrators must be able to run a Server Application that acts as FNAA.
Users using a Client Application acting as a FNUA must be able to:
1. Access the flow account and operate its flows. 2. Create a new flow. 3. Describe an existing flow. 4. Subscribe to an external flow.

6.2. FNAA - Server application

The FNAA server application must implement FNAP as described in Section 6. Basically, the FNAA will open a TCP port on all the IP addresses of the host to listen for new FNUA client connections.

The chosen language for the development of the FNAA is GoLang. The reason for choosing GoLang is because Kubernetes is written in this language and there is a robust set of libraries available for integration. Although there is no integration built with Kubernetes for this Proof of Concept, the usage of GoLang will enable a seamless evolution of the FNAA application. In future versions of the FNAA codebase, new functionalities leveraging Kubernetes will be easier to implement than if using a different programming language.

When the FNAA server application is initialized, it provides debug log messages describing all client interactions. In order to start the server application, a Network Participant system administrator can download the binary and execute it in a terminal:

```
ignatius ~ $ ./fnaad
server.go:146: Listen on [::]:61000
server.go:148: Accept a connection request.
```

Now that the 61000 TCP port is open, we can test the behaviour by means of a raw TCP using telnet command in a different terminal:
```
ignatius ~ $ telnet localhost 61000
Trying 127.0.0.1... Connected to localhost. Escape character is '^]'.
220 fnaa.unix.ar FNAA
```
We can now see that the server has provided the first message in the connection: a welcome message indicating its FQDN fnaa.unix.ar.

On the other hand, the server application starts providing debug information for the new connection established:
```
ignatius ~ $ ./fnaad
server.go:146: Listen on [::]:61000
server.go:148: Accept a connection request.
server.go:154: Handle incoming messages.
server.go:148: Accept a connection request.
```
6.3. FNUA - Client application

In order to test the FNAA server application, a CLI-based FNUA application has been developed. The chosen language for this CLI tool is also GoLang. The reason for choosing GoLang for the FNUA is because of its functionalities for building CLI tools, leveraging on the Cobra library. Thus, the FNUA for the PoC is an executable file that complies with the diagram in Figure 14.

One of the requirements for the flow CLI tool is a configuration file that defines the different FNAA servers together with the credentials to use. An example of this configuration file follows:

ignatius ~$ cat .flow.yml
agents:
  - name: fnaa-unix
    fqdn: fnaa.unix.ar
    username: test
    password: test
    prefix: unix.ar-
  - name: fnaa-emiliano
    fqdn: fnaa.emiliano.ar
    username: test
    password: test
    prefix: emiliano.ar-

namespaces:
  - name: flows.unix.ar
    agent: fnaa-unix
  - name: flows.emiliano.ar
    agent: fnaa-emiliano

In this file, we can see that there are two FNAA instances described with FQDN fnaa.unix.ar and fnaa.emiliano.ar. Then, there are two namespaces: one called flow.unix.ar hosted on fnaa-unix and second namespace flows.emiliano.ar hosted on fnaa-emiliano. This configuration enables the FNUA to interact with two different FNAA, each of which is hosting different Flow Namespaces.

Once the configuration file has been saved, the flow CLI tool can now be used. In the following sections, we will show how to use the minimum functionalities required for the Open Network using this CLI tool.
6.4. Use cases

### Use case 1: Authenticating a user

After the connection is established, the first command that the client must execute is the authentication command. As previously defined in Chapter 5, every FNAA client must first authenticate in order to execute commands. Thus, the authentication challenge must be supported both by the FNAA as well as the FNUA.

It is worth mentioning that the chosen authentication mechanism for this PoC is SASL Plain. This command can be extended furtherly with other mechanisms in later versions. However, this simple authentication mechanism is sufficient to demonstrate the authentication step in the FNAP.

The SASL Plain Authentication implies sending the username and the password encoded in Base64. The way to obtain the Base64 if we consider a user test with password test, is as follows:

```
ignatius ~ 0$echo -en "\0test\0test" | base64
AHRlc3QAdGVzdA==
```

Now, we can use this Base64 string to authenticate with the FNAA. First, we need to launch the FNAA server instance:

```
ignatius~/ $./fnaad --config ./fnaad_flow.unix.ar.yaml
main.go:41: Using config file: ./fnaad_flow.unix.ar.yaml
main.go:57: Using config file: ./fnaad_flow.unix.ar.yaml
server.go:103: Listen on [::]:61000
server.go:105: Accept a connection request.
```

Then, we can connect to the TCP port in which the FNAA is listening:

```
ignatius ~ 1$telnet localhost 61000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 fnaa.unix.ar FNAA
AUTHENTICATE PLAIN
220 OK
AHRlc3QAdGVzdA==
220 Authenticated
```

Once the client is authenticated, it can start executing FNAP commands to manage the Flow Namespace of the authenticated user. For simplicity purposes, in this Proof of Concept, we will be using a single user.
In the case of the CLI tool, there is no need to perform an authentication step, since every command the user executes will be preceded by an authentication in the server.

6.4.1. Use case 2: Creating a flow

Once the authentication is successful, the client can now create a new Flow. The way to do this using the CLI tool would be:

```
ignatius ~:/ 0$./fnua create flow time.flow.unix.ar
Resolving SRV for fnaa.unix.ar. using server 172.17.0.2:53
Executing query fnaa.unix.ar. IN 33 using server 172.17.0.2:53
Executing successful: [fnaa.unix.ar. 604800 IN SRV 0 0 61000 fnaa.unix.ar.]
Resolving A for fnaa.unix.ar. using server 172.17.0.2:53
Executing query fnaa.unix.ar. IN 1 using server 172.17.0.2:53
Executing successful: [fnaa.unix.ar. 604800 IN A 127.0.0.1]
Resolved A to 127.0.0.1 for fnaa.unix.ar. using server 172.17.0.2:53
C: Connecting to 127.0.0.1:61000
C: Got a response: 220 fnaa.unix.ar FNAA
C: Sending command AUTHENTICATE PLAIN
C: Wrote (20 bytes written)
C: Got a response: 220 OK
C: Authentication string sent: AHRlc3QAdGVzdA==
C: Wrote (18 bytes written)
C: Got a response: 220 Authenticated
C: Sending command CREATE FLOW time.flows.unix.ar
C: Wrote (31 bytes written)
C: Server sent OK for command CREATE FLOW time.flows.unix.ar
C: Sending command QUIT
C: Wrote (6 bytes written)
```

The client has discovered the FNAA server for Flow Namespace flow.unix.ar by means of SRV DNS records. Thus, it obtained both the FQDN of the FNAA together with the TCP port where it is listening, in this case 61000. Once the resolution process ends, the FNUA connects to the FNAA. First, the FNUA authenticates with the FNAA and then it executes the create flow command.

If we were to simulate the same behavior using a raw TCP connection, the following steps would be executed:
```
ignatius ~:/ 1$telnet localhost 61000 Trying 127.0.0.1... Connected to localhost. Escape character is '^[']'. 220 fnaa.unix.ar FNAA AUTHENTICATE PLAIN 220 OK AHRlc3QAdGVzdA== 220 Authenticated CREATE FLOW time.flows.unix.ar 220 OK time.flows.unix.ar ~
```
Now, the client has created a new flow called time.flows.unix.ar located in the flows.unix.ar namespace. The FNAA in background has created a Kafka Topic as well as the necessary DNS entries for name resolution.

6.4.2. Use case 3: Describing a flow

Once a flow has been created, we can obtain information of it by executing the following command using the CLI tool:

```
ignatius ~/ 1$./fnua describe flow time.flow.unix.ar
Resolving SRV for fnaa.unix.ar. using server 172.17.0.2:53
Executing query fnaa.unix.ar. IN 33 using server 172.17.0.2:53
Executing successful: [fnaa.unix.ar. 604800 IN SRV 0 0 61000 fnaa.unix.ar.]
Nameserver to be used: 172.17.0.2
Resolving A for fnaa.unix.ar. using server 172.17.0.2:53
Executing query fnaa.unix.ar. IN 1 using server 172.17.0.2:53
Executing successful: [fnaa.unix.ar. 604800 IN A 127.0.0.1]
Resolved A to 127.0.0.1 for fnaa.unix.ar. using server 172.17.0.2:53
C: Connecting to 127.0.0.1:61000
C: Got a response: 220 fnaa.unix.ar FNAA
C: Sending command AUTHENTICATE PLAIN
C: Wrote (20 bytes written)
C: Got a response: 220 OK
C: Authentication string sent: AHRlc3QAdGVzdA==
C: Wrote (18 bytes written)
C: Got a response: 220 Authenticated
C: Sending command DESCRIBE FLOW time.flow.unix.ar
C: Wrote (33 bytes written)
C: Server sent OK for command DESCRIBE FLOW time.flow.unix.ar
Flow time.flow.unix.ar description:
  flow=time.flow.unix.ar
  type=kafka
  topic=time.flow.unix.ar
  server=kfl.unix.ar:9092
Flow time.flow.unix.ar described successfully
Quitting
C: Sending command QUIT
C: Wrote (6 bytes written)
```

In the output of the describe command we can see all the necessary information to connect to the Flow called time.flow.unix.ar: (i) the type of Event Broker is Kafka, (ii) the Kafka topic has the same name of the flow and (iii) the Kafka Bootstrap server with port is provided. If we were to obtain this information using a manual connection, the steps would be:
ignatius ~ 1$telnet localhost 61000
Trying 127.0.0.1...
Connected to localhost.
Escape character is ‘^]’.
220 fnaa.unix.ar FNAA
AUTHENTICATE PLAIN
220 OK
AHRlc3QAdGVzdA==
220 Authenticated
DESCRIBE FLOW time.flows.unix.ar
220 DATA
flow=time.flows.unix.ar
type=kafka
topic=time.flows.unix.ar
server=kf1.unix.ar:9092
220 OK

Now, we can use this information to connect to the Kafka topic and start producing or consuming events.

6.4.3. Use case 4: Subscribing to a remote flow

In this section, we will show how a subscription can be set up. When a user commands the FNAA to create a new subscription to a remote Flow, the local FNAA server first needs to discover the remote FNAA server. Once the server is discovered by means of DNS resolution, the local FNAA contacts the remote FNAA, authenticates the user and then executes a subscription command.

Thus, the initial communication between the FNUA and the FNAA, in which the user indicates to subscribe to a remote flow, would be as follows: ˜˜ ignatius ~ 1$telnet localhost 61000 Trying 127.0.0.1...
Connected to localhost. Escape character is ‘^]’.
220 fnaa.unix.ar
FNAA AUTHENTICATE PLAIN 220 OK AHRlc3QAdGVzdA==
220 Authenticated
SUBSCRIBE time.flows.unix.ar LOCAL emiliano.ar-time.flows.unix.ar
220 OK

Once the user is authenticated, a SUBSCRIBE command is executed. This command indicates first the remote flow to subscribe to. Then, it also specifies with LOCAL the flow where the remote events will be written. In this example, the remote flow to subscribe to is time.flows.unix.ar, and the local flow is emiliano.ar-time.flows.unix.ar. Basically, a new flow has been created, emiliano.ar-time.flows.unix.ar, where all the events of flow time.flows.unix.ar will be written.
The server answers back with a new Flow URI, in this case ksdj898.time.flows.unix.ar. This Flow URI indicates a copy of the original flow time.flows.unix.ar created for this subscription. Thus, the remote FNAA has full control over this subscription, being able to revoke it by simply deleting this flow or applying Quality of Service rules.

The remote FNAA has set up a Bridge Processor to transcribe messages in topic time.flows.unix.ar to the new topic ksdj898.time.flows.unix.ar. Another alternative to a Bridge Processor would be a Distributor Processor, which could be optimized for a Flow with high demand. Moreover, instead of creating a single Bridge Processor per subscription, a Distributor Processor could be used, in order to have a single consumer of the source flow and write the events to several subscription flows.

The user could use the FNUA CLI tool to execute this command in the following manner:
ignatius ~ 0$./fnua --config=./flow.yml subscribe time.flows.unix.ar --nameserver 172.17.0.2 -d --agent fnaa-emiliano

Initializing initConfig
  Using config file: ./flow.yml

Subscribe to flow
Agent selected: fnaa-emiliano

Resolving FNAA FQDN fnaa.emiliano.ar
Starting FQDN resolution with 172.17.0.2
Resolving SRV for fnaa.emiliano.ar. using server 172.17.0.2:53
Executing query fnaa.emiliano.ar. IN 33 using server 172.17.0.2:53
FNAA FQDN Resolved to fnaa.emiliano.ar. port 51000
Resolving A for fnaa.emiliano.ar. using server 172.17.0.2:53
Resolved A to 127.0.0.1 for fnaa.emiliano.ar. using server 172.17.0.2:53
C: Connecting to 127.0.0.1:51000
C: Got a response: 220 fnaa.unix.ar FNAA
Connected to FNAA

Authenticating with PLAIN mechanism
C: Sending command AUTHENTICATE PLAIN
C: Wrote (20 bytes written)
C: Got a response: 220 OK
C: Authentication string sent: AHRlc3QAdGVzdA==
C: Wrote (18 bytes written)
C: Got a response: 220 Authenticated

Authenticated

Executing command SUBSCRIBE time.flows.unix.ar LOCAL emiliano.ar-time.flows.unix.ar
C: Sending command SUBSCRIBE time.flows.unix.ar LOCAL emiliano.ar-time.flows.unix.ar
C: Wrote (67 bytes written)
C: Server sent OK for command SUBSCRIBE time.flows.unix.ar LOCAL emiliano.ar-time.flows.unix.ar
Flow emiliano.ar-time.flows.unix.ar subscription created successfully
Server responded: emiliano.ar-time.flows.unix.ar SUBSCRIBED TO ksdj898.time.flows.unix.ar

Quitting
C: Sending command QUIT
C: Wrote (6 bytes written)
Connection closed

This interaction of the FNUA with the FNAA of the Flow Namespace emiliano.ar (fnaa-emiliano) has trigged an interaction with the FNAA of unix.ar Flow Namespace (fnaa-unix). This means that before fnaa-emiliano was able to respond to the FNUA, a new connection was opened to the remote FNAA and the SUBSCRIBE command was executed.

The log of fnaa-emiliano when the SUBSCRIBE command was issued looks as follows:
server.go:111: Handle incoming messages.
server.go:105: Accept a connection request.
server.go:253: User authenticated
flows.unix.ar
server.go:401: Flow is REMOTE
client.go:280: **#Resolving SRV for time.flows.unix.ar. using server 172.17.0.2:5
3
server.go:417: FNAA FQDN Resolved to fnaa.unix.ar. port 61000
client.go:42: C: Connecting to 127.0.0.1:61000
client.go:69: C: Got a response: 220 fnaa.unix.ar FNAA
server.go:435: Connected to FNAA
server.go:436: Authenticating with PLAIN mechanism
client.go:126: C: Sending command AUTHENTICATE PLAIN
client.go:133: C: Wrote (20 bytes written)
client.go:144: C: Got a response: 220 OK
client.go:154: C: Authentication string sent: AHRlc3QAdGVzdA==
client.go:159: C: Wrote (18 bytes written)
client.go:170: C: Got a response: 220 Authenticated
server.go:444: Authenticated
client.go:82: C: Sending command SUBSCRIBE time.flows.unix.ar
client.go:112: C: Server sent OK for command SUBSCRIBE time.flows.unix.ar
server.go:456: Flow time.flows.unix.ar subscribed successfully
server.go:459: Quitting

We can see how fnaa-emiliano had to trigger a client subroutine to contact the remote fnaa-unix. Once the server FQDN, IP and Port is discovered by means of DNS, a new connection is established and the SUBSCRIBE command is issued. Here we can see the log of fnaa-unix:

server.go:111: Handle incoming messages.
server.go:105: Accept a connection request.
server.go:253: User authenticated
server.go:139: Received command: subscribe
server.go:348: [SUBSCRIBE time.flows.unix.ar]
server.go:367: Creating flow endpoint time.flows.unix.ar
server.go:368: Creating new topic ksdj898.time.flows.unix.ar in Apache Kafka instance kafka_local
server.go:369: Creating Flow Processor src=time.flows.unix.ar dst=ksdj898.time.flows.unix.ar
server.go:370: Adding DNS Records for ksdj898.time.flows.unix.ar
server.go:372: Flow enabled ksdj898.time.flows.unix.ar
server.go:139: Received command: quit

Thus, we were able to set up a new subscription in fnaa-emiliano that trigger a background interaction with fnaa-unix.
6.5. Results of the PoC

We can confirm the feasibility of the overall Event Streaming Open Network architecture. The test of the proposed protocol FNAP and its implementation, both in the FNAA and FNUA (CLI application), show that the architecture can be employed for the purpose of distributed subscription management among Network Participants.

The minimum functionalities defined both for the Network Participants and the Users were met. Network Participants can run this type of service by means of a server application, the FNAA server. Also, the CLI-tool resulted in a convenient low-level method to interact with a FNAA server.

In further implementations, the server application should be optimized as well as secured, for instance with a TLS handshake. Also, the CLI-tool could be enhanced by a web-based application with a friendly user interface.

Nevertheless, the challenge for a stable implementation of both components is the possibility of supporting different Event Brokers and their evolution. Not only Apache Kafka should be supported but also the main Public Cloud providers events solutions, such as AWS SQS or Google Cloud Pub/Sub. Since the Event Brokers are continuously evolving, the implementation of the FNAA component should keep up both with the API and new functionalities of these vendors.

Regarding the protocol design, it would be needed to enhance the serialization of the exchanged data. In this sense, it could be convenient to define a packet header for the overall interaction between the FNAA both with remote FNAA as well as with FNUA.

Regarding the subscription use case, it would be necessary to establish a convenient format for the server response. Currently, the server is returning a key/value structure with the details of the Flow. This structure may not be the most adequate, since it may differ depending on the Event Broker used.

Also, the security aspect needs further analysis and design since its fragility could lead to great economical damage for organizations. Thus, it would be recommended to review the different security controls needed for this solution as part of an Information Security Management System.
Finally, the implementation should leverage the Cloud Native functionalities provided by the Kubernetes API. For example, the FNAA should trigger the deployment of Flow Processors on demand, in order to provide isolated computing resources for each subscription. Also, a Kubernetes resource could be developed to use the kubectl CLI tool for management, instead of a custom CLI tool.

7. Summary & Conclusions

In this chapter we will provide a summary of everything that has been described in this document as well as some conclusions about it.

We have identified a use case for which there is currently no adequate solution provided by existing tools. This use case is based on the cross-organization integration of real-time event streams. Nowadays, organizations intending to integrate these kind of data streams struggle with offline communication to achieve a common interface for integration. In this context, we proposed an Open Network for Event Streaming as a possible solution for this difficulty.

For this Open Network, we have followed the main necessities from the technical perspective. While there already exist many components that can be leveraged, some components require analysis, design, and implementation. Then, we referred to the Commons Infrastructure literature in order to show how Event Streaming can be considered an Infrastructure Resource that can enable downstream productive activities. Finally, we established the main guidelines that an Open Network should follow, basing these definitions on Free, Open & Neutral Networks.

Using the previous definitions, we have designed an architecture for the Event Streaming Open Network, establishing the components that the different Network Participants should implement in order to participate in the network. After providing a thorough description of all the components, we showed some use cases of integration among different Network Participants.

Once the architecture was defined, we proposed an implementation approach which describes the existing components that can be leveraged as well as those that need to be developed from scratch. The outcome was that a server-side application called FNAA had to be developed. This application implements the protocol FNAP and can be accessed by a client application, which we named FNUA.
Finally, we proved the feasibility of the proposed architecture by providing an implementation of the minimum functionalities required, in the form of a Proof of Concept. The results of this PoC were encouraging since it was possible to implement the initial functionalities for the FNAA and FNUA components.

As conclusion, we can mention that there is great potential for an Open Network for Event Streaming among organizations. In the same way the email infrastructure acts as an open network for electronic communications among people, this kind of network would enable developers to integrate real-time event streams while minimizing offline agreement of interfaces and technologies.

However, there are many difficulties that could be furtherly worked on. First, a robust implementation for the Event Streaming Open Network main components must be provided, mainly for the FNAA and FNUA. In order to achieve an acceptable level of quality and stability, the development of a community around the project is needed.

Secondly, we found that the proposed architecture is a convenient starting point. However, it can suffer modifications based on the learning process during the implementation. For example, while designing the architecture, we avoided the need of a database for the FNAA component, leveraging on the DNS infrastructure. While this can be sufficient for the minimum functionalities described, it will most probably be necessary for the FNAA to persist data in a database of its own. In this sense, we believe that leveraging the Kubernetes resources model could be a convenient alternative.

Thirdly, during the PoC execution, we identified some difficulties implementing the security functionalities of authentication and authorization. Although we were able to implement an authentication mechanism, the reality indicates that integration with well-established protocols is needed (i.e., OAuth, GSSAPI, etc.).

Finally, there is also the need to leverage on the Cloud Native architecture, basically Kubernetes, to provide hyper-scalability and enable Network Participants to agnostically choose the underlaying infrastructure. The selection of Golang for the PoC implementation showed to be convenient, given the vast number of available libraries for integration of third-party components and services.

Notwithstanding the difficulties, we firmly believe that cross-organization real-time event integration can provide great benefits for society. It would enhance the efficiency of business processes throughout organizations. Also, it would provide broad visibility to the final users, enabling experimentation and entrepreneurship.
business models for existing productive activities could be
developed, as well as enabling innovation, which in turn would
conform the positive externalities of the Event Streaming Open
Network.

8. Security Considerations

TODO Security

9. IANA Considerations

This document has no IANA actions.

10. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
        Requirement Levels", BCP 14, RFC 2119,
        DOI 10.17487/RFC2119, March 1997,

        specifying the location of services (DNS SRV)", RFC 2782,
        DOI 10.17487/RFC2782, February 2000,

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,
        A., Peterson, J., Sparks, R., Handley, M., and E.
        Schooler, "SIP: Session Initiation Protocol", RFC 3261,
        DOI 10.17487/RFC3261, June 2002,

        DOI 10.17487/RFC5321, October 2008,

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service
        Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013,

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
        2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,

Acknowledgments

SPINELLA E. (2022) [Online] Event Streaming Open Network Master’s
Thesis https://drive.google.com/file/d/1R9H-4knAztez_yUPlr7aZSkbUjs8jL3j

Spinella Expires 31 July 2022
URQUHART J. (2021) Flow Architectures


WIDL M. (2013), Guided Merging of Sequence Diagrams


GUTTRIDGE, Gartner (2021) "Modern Data Strategies for the Real-time Enterprise" Big Data Quarterly 2021

Author’s Address

Emiliano Spinella
Syndeno

Email: emiliano.spinella@syndeno.com