Internet Engineering Task Force
INTERNET-DRAFT t.b.d

Nicolas Zunker
TU Berlin
March 19, 2023

PVS: A Peer to Peer View Sampling Protocol

Abstract

This document describes the Peer to Peer View Sampling Protocol (PVS), an extendable protocol for peers within a peer sampling service to exchange views. The PVS protocol provides a general structure for the exchanging of views, without placing any restrictions on the address format of the peers, or what sort of metadata may be passed alongside them. The protocol also allows peer sampling services to define their own formats and can be used as the building block for peer sampling services.

Contents

1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

Peers, Peer to Peer Networks, and Peer Sampling Services : In this document the key word "PEER" refers to any host exchanging the messages specified here, via any transmission scheme (e.g. via the internet, a local socket, over a wire etc.) . The key word "PEER TO PEER NETWORKS" refers to networks made up of multiple peers, which are all executing the same application. Finally "PEER SAMPLING SERVICE" refers to a service, within a peer to peer network, for the discovery of other peers, via the exchange of views between peers.

Views : In this document the key work "VIEW", when used in the context of a peer, refers to that peer's internal list of the other peers, within the network, which it knows.

2. Introduction

The core building block of any peer sampling service is the exchanging of views. The decisions as to how those views are stored, how a peer is selected, and how views are merged, are up to the peer sampling service itself, but regardless of how those problems are solved, any peer sampling service requires a view exchange procedure. This document aims to provide an extensible message format for this kind of transaction, such that any peer sampling service may use this protocol for its view exchange transactions. Central to this goal is the abstraction of how an individual peer is stored within a view. Since different peer sampling services may want to store different data about the peers within their network, this protocol can be customized to support arbitrary formats for representing peers.

This document will be structured as follows: the rest of this section will describe why and how different peer sampling services can use this protocol, then the following sections will introduce the message formats and types of addresses and metadata, and to conclude, the transport protocol considerations will be discussed.

2.1. Custom Data Formats

Within the context of this protocol, how a peer, and views of peers, are represented will vary depending on the peer sampling service at hand. Since different peer sampling services may store different data about the peers in their views, this protocol prepends each peer "address" in a message with its address type. Based on this value, applications can determine what format the peer "address" is actually in (e.g. IPv4 address + Port + UTC timestamp).

Any peer address, or metadata associated with an address or a view exchange message, must specify what format it is in. The format is indicated with an integer that corresponds to a message format either specified here, or known to the nodes of the application. This allows developers to use the message formats from this document if they wish to, but also enables them to create new formats to cater to the needs of their own peer sampling service.

There are two classes of codes for data formats. This first class of codes must correspond to publicly known and specified data formats, such as the ones that will be introduced here. These codes are called "General Codes". All protocols or peers sampling services which use the message formats from this document must respect the general codes and only use them in conjunction with the data format which they are specified for. Crucially, General codes will only be specified once. This document will define and specify a few initial General Codes, to transport commonly used data types.

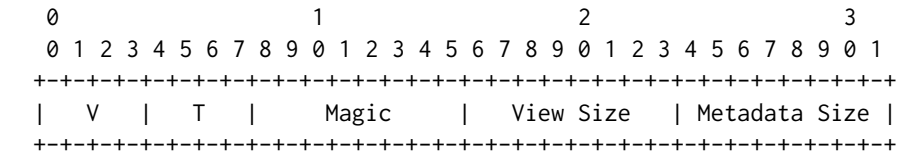
The second class of codes are "Application Specific Codes". These codes may be specified in public documents or may be used privately by an application. Most importantly, these codes may be reused, so that different peer sampling services can define their own custom data formats that may or may not have the same identifier. This document will open up an initial 64 Application Specific codes which are not specified and can be freely used by any application building upon this protocol.

3. Message Formats

At present there is only one message format, and only two types of messages. All messages must be view exchange messages, and when a node initiates a view exchange message with another node it must mark its message as being a request, and the response must also be marked as a response message.

3.1. View Exchange Messages

At its most basic level a view exchange message is composed of 3 components: the number of peers in the view, the view itself, and any optional metadata. To this end, the format of a View Exchange Message is centered around these 3 components. In the following figure the format of a view exchange message is shown. It consists of a version, a message type, a view size, the view, and message metadata. The Peer and Metadata formats will be specified in later subsections.



[Peers in View]
(Message Metadata)

Version of PVS (V), 4 bits:
This document specifies version 1 of PVS. Any implementation following this document MUST set this field to 1. Nodes which receive a message containing a different version MAY continue to attempt to process the message, but are RECOMMENDED to discard it.

Type of Message (T), 4 bits:
This field shall indicate the message type, as of now there are just two message types. Request (0) and Response (1). A node which initiates a view exchange procedure with another node MUST set this field to 0. When that node responds it MUST set this value to 1. Nodes which receive a message with a value other than 0 MAY continue to try processing it, but are RECOMMENDED to discard it.

Magic, 8 bits: This field should be set to the magic value 177 to give nodes an added level of security that the message with follows is a valid message conforming to this specification.

View Size, 8 bits:
This field indicates the size of the view, in terms of number of peers- not in bytes. This should be the number of entries in the list of peer addresses contained in the message body. Note that a view size of 0 or 1 is perfectly normal and will often be used by peers which have just joined the network.

Metadata Size, 8 bits:
This field indicates the number of metadata blocks which will follow the View. If this is 0 then the message contains no metadata blocks and the message ends after the View.

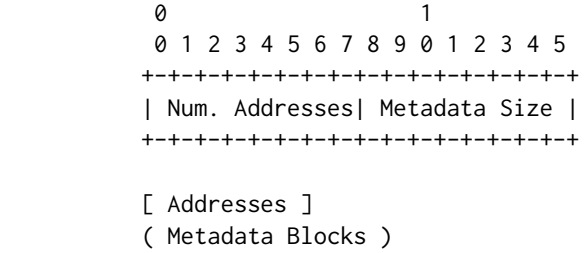
Peers in View:

This part of the message is a list of peer addresses. The number of entries in the list is given by the View Size field. Each entry is in the form of a Peer Block, as specified in subsection 3.1.1.

Metadata:
This part of the message is the optional metadata. It must be in the format of a Metadata Block, as is specified in section 3.1.3. If there is no metadata then the Metadata Size field MUST be 0.

3.1.1. Peer Blocks

Peer blocks represent an entry for a single peer in the view. For a single entry there may be multiple addresses and multiple metadata blocks. The format of a peer block is as follows:



Num. Addresses, 8 bits:
This field indicates the number of addresses which will follow. Each of these addresses MUST be a different means of reaching the given peer.

Metadata Size, 8 bits:
This field indicates the number of metadata blocks which will come after the Addresses.

Addresses:
A list of addresses for this peer, in the format of Address Blocks, as specified in section 3.1.2.

Metadata Blocks:
A list of metadata blocks. These must be in the format specified in section 3.1.3

3.1.2. Address Blocks

Address Blocks form the basis of this protocol. They represent a means of reaching a peer within the peer to peer network. Since peer sampling services may represent peers in different ways or may want to pass along metadata with a peer address there is no one size fits all solution. As such the format of a peer address should be variable. Therefore the Address block specifies the type of the address as well its length in bytes. Address blocks make use of the VarU64 (https://github.com/AljoschaMeyer/varu64) variable length encoding (the first byte indicates the length, if its < 248, otherwise it indicates that the length is encoded as an unsigned integer of [value of first byte - 247] number of bytes). The structure of an address block is as follows:

[Type]
[Address Length (Variable Length Encoded according to VarU64)]
[Raw Address Data]

Type, 8 bits:
This field indicates the format of the address which will follow. If a node does not recognize the address type it MUST skip this peer address (using the length field), and move on to the next address.

Address Length:
The length, in bytes, of the address which will follow, VarU variable length encoded. This field will be between 1 and 9 bytes long. If the value of the first byte in this field is less than 248 then the value itself is the length. If it is greater, then it determines how many of the following bytes are used to represent the value (in network byte order):

First Byte Value	Additional Bytes
248	1
249	2
250	3
251	4
252	5
253	6
254	7
255	8

Raw Peer Address Data: This is where the bytes that represent the peer address go. For example, if the type was 1 (IPv4 Address) then this would simply be 4 bytes representing the IPv4 address of a peer.

3.1.3. Metadata Blocks

Much like the Address Blocks, a Metadata Block is meant to be extensible to allow the transportation of any type of peer sampling service specific data. The metadata in a peer block should correspond to that peer. All messages in this protocol may also include a metadata block after the list of peers. This allows the sender to also pass along general data, not specific to any peer in the view. The format of a metadata block is essentially identical to that of an address block.

[Type]
[Metadata Length (Variable Length Encoded according to VarU64)]
[Raw Metadata]

Type, 8 bits: This field indicates the format of the metadata which will follow. If a node does not recognize the metadata type it MUST skip this block (using the length field).

Metadata Length:
The length of the address which will follow, in bytes. This field indicates the length of the metadata to follow, in bytes. For example if the metadata were of type 1 (UTC timestamp) its value would be 8. This field is variable length encoded according to VarU64, the same exact way as the corresponding Address Length field in the Address block section (section 3.1.2). Please see that section for how to

use the VarU64 encoding.

4. Address and Metadata Types for the General Peer Sampling Service

This section will define some initial, general types of address and metadata. Since the goal of this specification is to provide a general framework on top of which other peer sampling services may be built, these types will not be very extensive and there will not be many of them. All of the types which are defined here belong to the classes of general codes, which are meant for public use.

4.1. Address Types

- 0 : **Reflective Address**
  This type has a length of 0, and it indicates to the receiver that the address which the sender wishes to convey is their own address, as it appears to the receiver. For example this could be the IP address and port that are on the packet. This address type is designed to help peers which suspect they are behind a NAT, or peers wishing to save the space it would take to store their own address in the message.

- 1 : **IPv4 Address**
  This type has a length of 4 bytes, and it indicates to the receiver that the address which the sender wishes to convey is in the form of a 4 byte IPv4 address in Network Byte Order (NBO).

- 2 : **IPv4 Address + Port**
  This type has a length of 6 bytes, and it indicates to the receiver that the address which the sender wishes to convey is in the form of a 4 byte IPv4 address in NBO followed by a 2 byte Port number in NBO.

- 3 : **IPv6 Address**
  This type has a length of 16 bytes, and it indicates to the receiver that the address which the sender wishes to convey is in the form of a 16 byte IPv6 address in NBO.

- 4 : **IPv6 Address + Port**
  This type has a length of 18 bytes, and it indicates to the receiver that the address which the sender wishes to convey is in the form of a 16 byte IPv6 address in NBO followed by a 2 byte Port number in NBO.

As of now the remaining IDs $<= 127$ are reserved for future use as general codes, and those codes > 127 are free for use as application specific codes.

4.2. Metadata Types

- 0 : **Logical Timestamp**
  This type has a length of 4 bytes, and is a simple counter, represented as an unsigned integer in NBO. How this type should be used, interpreted, or incremented by a given Peer Sampling Service (PSS) is not within the scope of this specification, as these are choices that the PSS must make for itself.

- 1 : **UTC Timestamp**
  This type has a length of 8 bytes, and it should be a 64 bit signed integer indicating the the time as the number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 .

As of now the remaining IDs $<= 127$ are reserved for future use as general metadata codes, and those codes > 127 are free for use as application specific metadata codes.

5. Transport Protocol Considerations

Due to its simple request-response format, and lack of a concept of sessions, this protocol can work over most types of data communication channels- not just over the internet. However special attention will be given to IP networks since this is the most likely use case.

With regards to transport protocol, both UDP and TCP may be used, with any port. It should be noted that UDP may be preferrable for most peer to peer applications, since NAT hole-punching is easier over UDP, but TCP may be a better choice if fragmentation may be an issue, or if the application needs TCP's support for congestion avoidance and/or flow control.

5.1. TCP

Since TCP has the concept of sessions but UDP does not, when a peer aborts the processing of a message (since it is either malformed or non-compliant) the TCP connection should also be closed, whereas over UDP the peer will simply not respond. When using TCP, two peers do not need to tear down the connection after the first request-response transaction. The peer which initiated the TCP connection should always mark the first view exchange message as a request. For multiple transactions over the same TCP session, either peer can act as the initiator and start the transaction by sending a View Exchange Request. Otherwise, a peer may close the TCP connection after successful completion of a Request-Response exchange, and both peers can consider the transaction successful, and may initiate a new transaction over a new TCP connection at a later point, if desired.

5.2. UDP

For UDP the request and response View Exchange messages SHOULD map to a single UDP packet each, but dont need to. It is HIGHLY RECOMMENDED that the size of the UDP packets not exceed the path MTU, so as to avoid any issue with fragmentation (e.g. firewalls dropping packets, and/or the increased risk of losing a fragment and thus the message). Since peer sampling services (PSSes) should have a good idea of what kinds of address and metadata formats they support, they should be able to attempt to predict the worst case size of a view exchange message. Whether or not this exceeds the path MTU , can inform the decision of the PSS to use UDP or not. It is recommended that PSSes which intend to run over UDP and may occasionally or routinely exceed the MTU, define their own types of metadata and procedures which can use the metadata information to aid the receiving peer in reassembling messages which spawn multiple UDP packets.

With regards to flow control, the PSS must also consider whether or not this could become and issue and may need to implement a mechanism for solving this (e.g. a metadata type for indicating to the sender that they should speed up or slow down). However, generally, one would imagine that a PSS would only perform peer sampling periodically, and at a reasonable rate (reasonable meaning a rate that is unlikely to lead to congestion of the network, or overloading of receivers). Ultimately the decision whether or not to use UDP instead of TCP is a decision that peer sampling services must make for themselves. They must consider the dangers of fragmentation, if their messages routinely exceed the path MTU, as well as the fact that there is no inherent support for congestion and flow control. Peer sampling services which define large view sizes with complex address and/or metadata formats are much more likely to need to use TCP (for the increased message size and congestion control). Services which have high data rates, for whatever reason, will have to face the decision of whether or not to use UDP's inherent speed at the of risk overloading slow receivers, or using TCP's flow control but sacrificing some speed.

Finally, it should be noted that thanks to the VarU64 encoding it is possible to send incredibly large datagrams ($> 65535 * (2^{64} - 1)$ bytes if sending the maximum number of peers and metadata blocks, each of maximum size), which would lend itself to IPv6 jumbograms. In the case of IPv6 jumbograms, the measures mentioned in the corresponding specification [1] should be taken into consideration by the application.

5.3. Other Transport Protocols

For other protocols, such as QUIC or SCTP, the same considerations as above apply.

In any stream or FIFO-type transmission protocol a party must close the stream on reception of a bad message, and either party may send view exchange requests initiating new transactions, not just the party which opened the stream. Closing a stream after one transmission or leaving it open for multiple ones are both allowed.

For transmission protocols which do not provide a FIFO guarantee, the view exchange request and response messages should map to one packet, or the application must include its own logic to handle instances where this is not the case. Exceeding the path MTU is highly advised against, but up to the peer sampling service using this protocol.

Finally in the absence of flow control and/or congestion control in the underlying transport protocol, the peer sampling service should define its own means to avoid and/or deal with this.

[1] https://datatracker.ietf.org/doc/html/rfc2675