Hot Protocol Version 2 (HotPv2)

Hottentot RPC Framework

Kamran Amini

June 19, 2016

Contents

1 Introduction

This document talks about request and response structures and mechanisms in Hottentot RPC Framework. Purpose of this protocol is to convey Method Invocation request and response. Current protocol is serialization transparent and can convey a method call with arguments produced with different serialization algorithms. In this version, Hottentot's runtimes can only work with internal serialization mechanism.

2 Glossary

SERIALIZATION
A process in which an object turns into a byte array to be transferred using a channel.

STRUCT
A term used for encapsulation of fields related to a specific entity. It is a structure and it will be generated differently for each programming language.

IDL (INTERFACE DEFINITION LANGUAGE)
An IDL is a language transparent to all programming languages which Hottentot supports. IDL can be generated to any target languages supported by Hottentot RPC Framework.

HOT FILE
A file which contains IDL. Hot files usually end with `.hot` extension.

GENERATOR
A tool for generating stub and struct source codes for a target programming language. Currently, generators for C++ and Java languages are available.

RUNTIME
A library for a specific programming language which performs Service and Proxy operations. Currently, runtimes are only available for C++ and Java.

ENDPOINT
Endpoint is a combination of IP address and a port. One service object or many can be bound to an endpoint.

CURRENT ENDPOINT
When talking about a proxy, Current Endpoint is an endpoint which client has used for connecting to server-side.

SERVICE
Service is an object serving method invocation requests.

PROXY
Proxy is an object which produces method invocation and other types of requests and receives the response. A proxy object talks to an endpoint at first and its request will be delegated to a service object in case of method invocation requests.

PROXY-SIDE
A software or library which tries to interact with service-side objects like endpoints, services, etc.

SERVICE-SIDE
A software or library which serves proxy-side requests and generates suitable response.

PUBLIC KEY INFRASTRUCTURE (PKI)
An infrastructure built on top of specific asymmetric cryptography mechanisms and schemes to provide security solutions to known security issues.

3 Protocol V2

Protocol version 2 introduces new features to protocol version 1 in many aspects. Following sections talk about differences and new features in this version.

3.1 Connection Properties

Hottentot connections uses Session layer and Application layer in OSI model. Hottnetot SHOULD use *Secure Socket Library* for *Authentication* and *Encryption* purposes.

3.1.1 Transport

Hottentot SHOULD support TCP for its transport protocol. That's because of long payloads which need sequencing feature in transport layer. Hottentot MAY work with other transport protocols offering sequencing feature and this MAY be up to operating system to handle transport operations.

3.1.2 Type of Communication

Like version 1, communication in protocol version 2 starts with a request from proxy-side. In this version, server-side can not start the communication yet. This would be a future feature with its limitations. KeepAlive feature is not supported and connection SHOULD be closed by server-side after sending the response. All Hottentot connections SHOULD start with **handshake** in this version. Details about the handshake will be discussed in future chapters.

3.1.3 Error Detection

There is no error detection mechanism in this version. We trust underlying protocols like TCP for detecting errors found in the transimtted data. It's obvious that using some security features like SSL provides some tools for detection of communication errors but it is OPTIONAL.

3.1.4 Security

Default behaviour SHOULD use plain transfer of data. This means that security features are optional and they should be enabled by user using configurations. Following sections talk about different aspects of security concerns regarding the Hottentot communication. It is clear that default behaviour does not prevent none of following issues.

3.1.4.1 Eavesdropping

In order to prevent eavesdropping, Hottentot SHOULD use *Encryption*. Symmetric encryption SHOULD be enough but key distribution phase SHOULD use asymmetric cryptography. Any means necessary for implementing the eavesdropping prevention mechanism SHOULD be used in this scenario; examples of such means are certificates, public and private key pairs, etc.

3.1.4.2 Injection

In order to prevent any external party to inject data in a communication channel, Hottentot SHOULD use *Digital Signature*. Digital Signatures SHOULD use PKI in order to provide certificates and key pairs for authenticated parties. Hottentot requests and responses MAY carry Digital Signature over request data to assure other parties that data is not changed while transferring.

3.1.4.3 Authentication

In order to provide authentication for parties, Hottentot SHOULD use asymmetric cryptography and PKI compliant certificates.

3.1.5 State Management

Communications SHOULD be stateless. States related to service objects SHOULD be retained from the request payload. Stateful objects are not supported in this version.

3.2 Handshake

As stated before, Hottentot communications SHOULD start with *Handshake* phase. Since connections SHOULD start from proxy-side, *Handshake Request* SHOULD be sent from proxy-side application. Notice that, this handshake is specific to Hottentot and has nothing to do with other possible handshakes in the system like SSL handshakes or TCP handshakes.

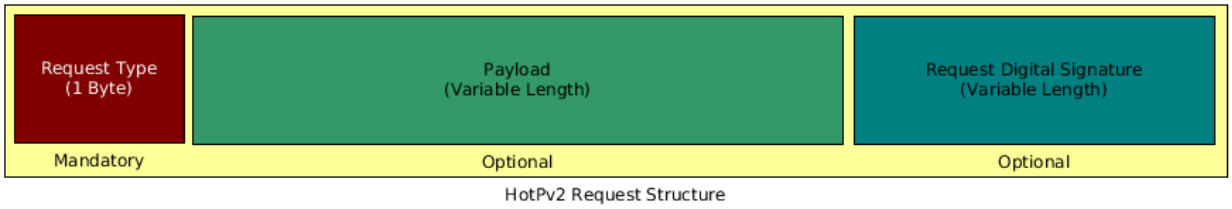
3.3 Request

Each request consists of 1 mandatory and 2 optional fields.

- **Request Type**: This field is mandatory. Using this field determines type of request.
- **Payload**: This files is optional and carries all necessary data to perform the request operation.
- **Request Digital Signature** This fields is optional and carries a PKI compliant digital signature over Request Type and Payload.

Please notice that, there is no checksum or error detection block involved in request structure. This means that we trust underlying network connection for handling communication errors. Also security concerns like impersonation, eavesdropping or on-wire injection will not be discussed here. There is a dedicated chapter for explaining security issues and detailed provided solutions. Below figure shows the request structure.

Figure 1: HotPv2 Request Structure



3.3.1 Request Type

Determines the request type and payload structure. Following C enumeration shows the possible values for this field.

```
enum RequestType {
  Unknown = 0x00,
  ServiceListQuery = 0x01,
  MethodInvocation = 0x02,
  MethodListQuery = 0x03,
  ServiceInfoQuery = 0x04,
  MethodInfoQuery = 0x05,
  EndpointInfoQuery = 0x06
};
```

Values can be:

- **Unknown**: It means nothing to Hottentot service side and these requests should be ignored by implementation.
- **ServiceListQuery**: Proxy-side queries about the list of available services. Hottentot service runtime should return list of services exposed on current endpoint.

- **MethodInvocation**: Invokes a method on a specific service object.
- **MethodListQuery**: Proxy-side queries about the list of callable methods on a specific service object.
- **ServiceInfoQuery**: Proxy-side asks about parameters of a specific service.
- **MethodInfoQuery**: Proxy-side asks about parameters of a specific method.
- **EndpointInfoQuery**: Proxy-side asks about parameters of current endpoint.

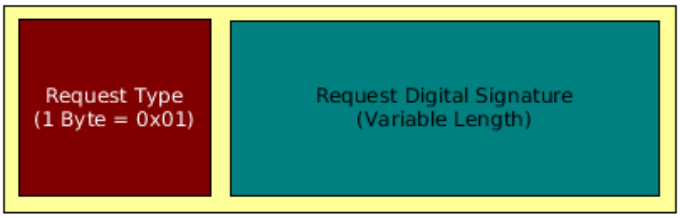
3.3.2 Payloads

Requests can have payloads. Many request types need data for their operations and payload carries the data. Following sections explain about payload structure for each request type. Please notice that, any needed piece of information or security object for authentication and authorization purposes will not be included in payloads and these objects will be transferred in *Handshake* phase.

3.3.2.1 Payload for *Service List Query* Request Type

A *Service List Query* request has no payload. In other words, this request has only one byte carrying value 0x01 as *Service List Query* request type identifier.

Figure 2: Structure of a Service List Query request.



3.3.2.2 Payload for *Method Invocation* Request Type

A *Method Invocation* request payload consists of following fields:

- Service Id (4 Bytes)
- Method Id (4 Bytes)
- Number of Arguments (1 Byte)
- Arguments as an array of LV Structures. (Variable Length)

Figure 3: HotPv2 Method Invocation Request Structure

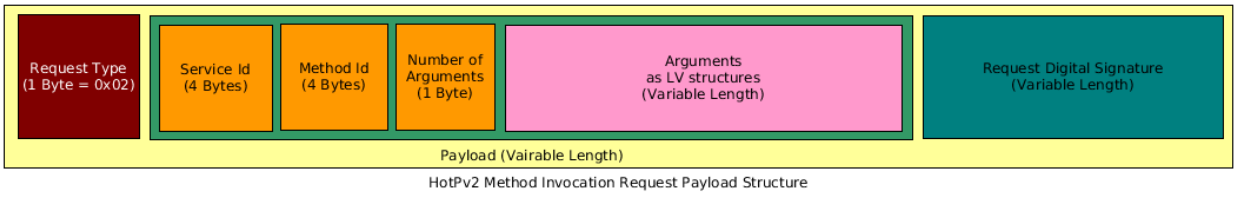
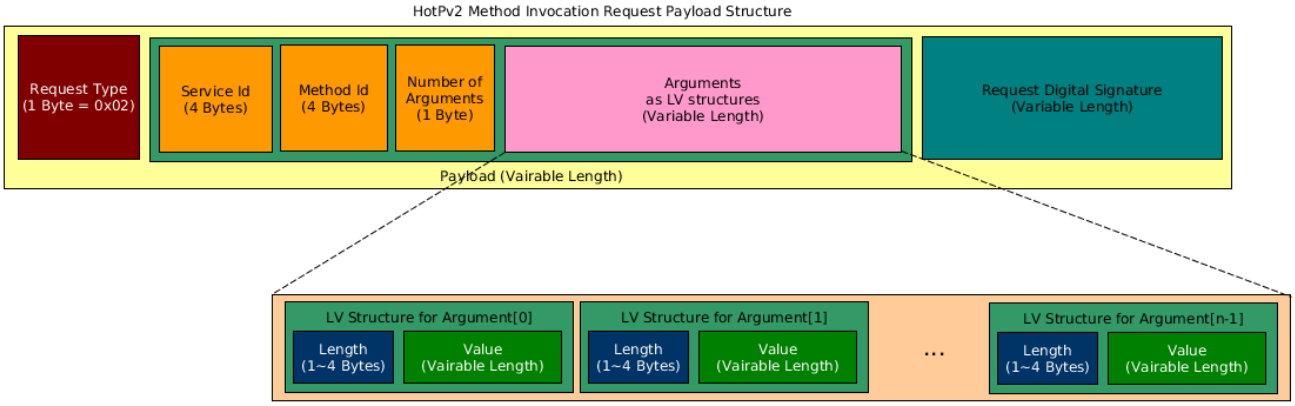


Figure 2 shows the structure of a request. Arguments can be transferred using LV structures. LV structures include Length and Value. Maximum length for a single LV structure is $2^{32} - 1$ since length values can be stored in a field at most 4 bytes. Figure 3 shows the LV structures in detail.

Value can be any byte array but usually it is a serialized object. Serialization method can be anything. Hottentot itself provides an algorithm for serialization and `serialize()` and `deserialize()` methods are generated for every struct. Current stub generation mechanism works only with Hottentot's internal serialization.

Figure 4: Structure of a Method Invocation request payload with arguments' LV structures.



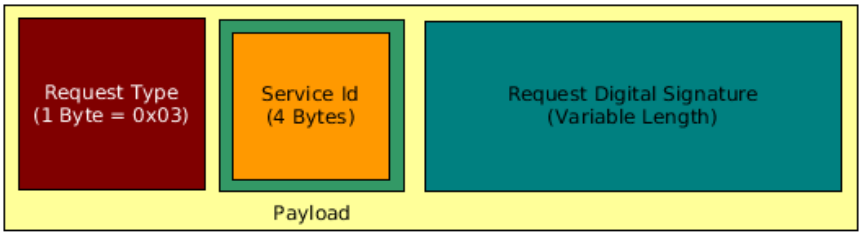
3.3.2.3 Payload for *Method List Query* Request Type

A *Method List Query* request has following fields:

- Service Id (4 Bytes)

This will return list of methods available on a service object identified by *Service Id*.

Figure 5: Structure of a Method List Query request.



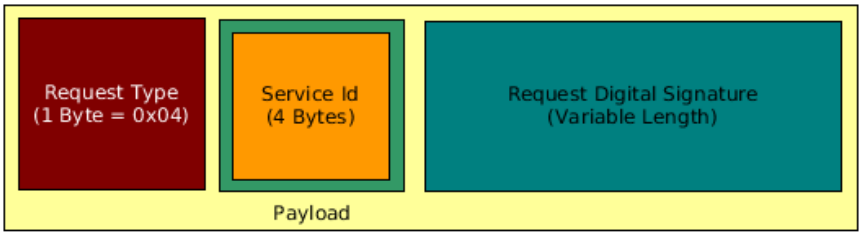
3.3.2.4 Payload for *Service Info Query* Request Type

A *Service Info Query* request has following fields:

- Service Id (4 Bytes)

This will return the parameters and their values for a service object identified by *Service Id*. Parameters and values are serialized according to LV structures. Structure for service parameters will be explained in future chapters.

Figure 6: Structure of a Service Info Query request.



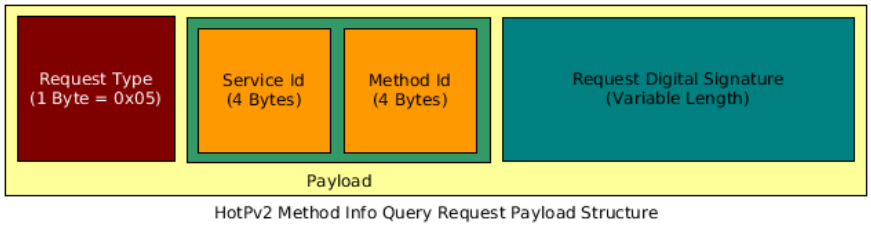
3.3.2.5 Payload for *Method Info Query* Request Type

A *Method Info Query* request has following fields:

- Service Id (4 Bytes)
- Method Id (4 Bytes)

This will return the parameters and their values for a method identified by *Method Id* on a specific service object identified by *Service Id*. Parameters and values are serialized according to LV structures. Structure for method parameters will be explained in future chapters.

Figure 7: Structure of a Method Info Query request.

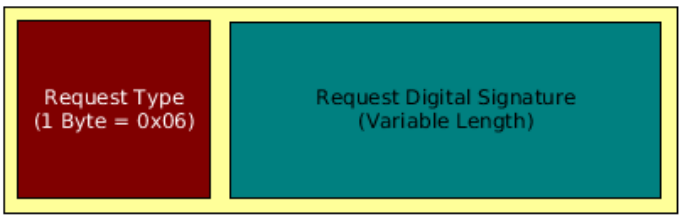


HotPv2 Method Info Query Request Payload Structure

3.3.2.6 Payload for *Endpoint Info Query* Request Type

A *Endpoint Info Query* request has no fields, hence we have no payload. Parameters of current endpoint selected by proxy-side will be returned as response. Parameters are serialized according to LV structures.

Figure 8: Structure of a Endpoint Info Query request.



HotPv2 Endpoint Info Query Request Payload Structure

4 Algorithms

4.1 Fast Hash Algorithm

A fast hashing algorithm developed by Zilong Tan. It has 2 versions. One version generates 32-bit has value for a given byte array and the other one generates a 64-bit hash value for the input. Hottentot uses 32-bit fast hash algorithm for generating system independent identifiers (ids) for services and methods. We denote this algorithm with FH_{32} and FH_{64} throughout this document.

$$\begin{aligned} FH_{32} &: \{0,1\}^* \rightarrow \{0,1\}^{32} \\ FH_{64} &: \{0,1\}^* \rightarrow \{0,1\}^{64} \end{aligned} \tag{1}$$

5 Future Features

- Version should be added to request and response structures.