Top (Type Oriented Protocol)

Pasqualino 'Titto' Assini (tittoassini@gmail.com)

22nd of May 2017

Contents

What is Top?

Top, the **T**ype **O**riented **P**rotocol, is a minimalist content-oriented transport protocol.

Purpose

Top aims to provide a practical solution to the fundamental mismatch that exists between the way information and services are commonly published on the Internet, grouped by their publisher, and the way information is most commonly requested by its prospective users, by subject or type.

For example, consider the obvious mismatch between the way information regarding flights is published, via airlines' specific websites, and the way users need it, by departure and destination airport, price etc.

This essential mismatch, that makes information and services hard to locate and share, is currently addressed by complex search engines and aggregation sites that ’scrape’ data from web sites and services, each typically organised and structured in different and incompatible ways.

This ’solution’ is deeply unsatisfying, expect for the few major commercial companies that, having developed the almost unimaginably complex search engines required to ’patch’ the broken way in which information is published, have become the oligopolistic gatekeepers of the Internet, arising substantial issues of commercial and political control.

Top, as other content-oriented protocols, aims to address the problem at its root by making the publishing of data ordered by type the default rather than the exception.

Though this in itself does not fully resolve the problem of effectively locating and sharing information, it goes a long way in facilitating and democratising information access.

Top allows data providers and data customers to, incrementally and organically, develop a shared dictionary of data type definitions that precisely specify the structure of data exchanged and the functionality of online services and to connect them via shared global typed communication channels.

Design Principles

The design follows the End-to-End principle, the protocol provides only absolutely essential functionality, additional features are provided by services and agents commuticating via the protocol.

Design Goals

- Consistent Expandibility: ability to implement any conceivable additional functionality over the basic protocol while using the same conceptual framework
- Independent Evolvability: ability of every agent to evolve the system independently, without relying on explicit consensus from other parties
- Transparency: ability to understand the syntactical structure and semantics of the information exchanged by all parties
- Scalability: ability to operate efficiently on a global scale, with an unlimited number of agents, exchanging an unlimited number of messages
- Minimalism: the absolutely minimum functionality required to satisfy the other design goals

Operating Model (normative)

Conceptually, all communication in *Top* takes place on a single global channel to which any communicating agent can anonymously send and from which it can anonymously receive *typed values*.

Top's delivery model consists in a single rule: every value sent by an agent SHOULD be faithfully delivered exactly once to every other agent currently connected.

In other words, *Top* adopts a best-effort delivery approach, it tries to provide but does not guarantee:

- Unicity: an agent receives exactly one copy of every value sent after its connection to the system
- Correctness: an agent never receives spurious values, values never sent by any agent

Top emphatically does not provide:

- Ordering: an agent receives values in the same order in which they were sent by the originating agent (intra-agent ordering) and in the chronological order in which they were sent by different agents (inter-agent ordering)
- Persistency: agents receive values that were sent before their connection to *Top*

Or indeed any other property not explicitly specified, such as error detection, data compression, determination of provenance, etc.

The expandibility of the protocol, however, allows all this additional functionality to be added on a as-required base.

Typed Values (normative)

Top transfers only serialised typed values.

Data types are defined according to the 正名 (Zhèng Míng) data modelling language.

Data values are serialised according to the Flat binary format.

Conceptually, the single global *Top* channel transfers values of type:

```
TypedBLOB ≡ TypedBLOB (Type AbsRef) (BLOB FlatEncoding)
```

A *TypedBLOB* has two components: a type and a serialised value of the specified type.

Routing Protocols

In practice, agents do not connect directly to the global channel.

Agents interacts with *Top* by opening one or more connections to one or more *Top* access points (AP), complying with a given routing protocol.

The list of supported routing protocols is open ended, APs are free to provide additional routing protocols.

Top does not directly provide a mechanism to discover an access point and/or their routing protocol capabilities.

ByPattern Routing Protocol (normative)

All APs SHOULD provide support for the ByPattern routing protocol.

The protocol provides a full-duplex communication channel that allows a connected agent to send values of a given type and to receive all values of the same type sent by any other agent that match a specified pattern.

`ByPattern` connections are described by values of type `ByPattern a` where `a` is the type of the values being transmitted:

```
ByPattern a ≡ ByPattern (List (Match (List Bit)))
```

```
Match a ≡   MatchValue a
          | MatchAny (Type AbsRef)
```

A value is matched if its serialisation is completely matched by the list of specified matches.

A match can be:

- a `MatchValue (List Bit)` that matches the specified sequence of bits
- or a `MatchAny (Type AbsRef)` that matches any serialised value of the indicated type

CHAnnel ***T***ype ***S***election Protocol (normative)

`CHATS` is a simple and universal way for an agent to request a communication channel that complies with a desired high-level protocol.

The actors are an agent and an access point (AP) accessible via a known full-duplex transport protocol at a known address.

The protocol operates as follows.

The agent opens a connection channel to the AP using one of the transport protocols supported by both AP and agent.

Once the connection has been established according to the usual conventions of the underlying transport protocol, the agent sends on the channel the `Flat` encoding of a value of type `TypedBLOB` specifying the desired communication protocol.

The AP replies by sending to the agent a value of type:

```
ChannelSelectionResult a ≡   Success
                           | Failure {reason :: List Char}
                           | RetryAt a
```

where *a* is the type corresponding to the transport protocol used to connect to the AP (for example: `WebSocketAddress IP4Address`).

If the AP is able and willing to switch the channel to the requested routing protocol, it will reply with `Success` and from this moment on communication on the channel takes place according to the selected routing type till either the AP or the agent proceed to close the channel according to the usual conventions of the underlying transport protocol.

Otherwise, if the AP is aware of a different AP that might provide the required protocol, it will return a value `Retry address` and it will close the channel. The agent can then try to connect to the Retry address. To avoid infinite redirection loops, the agent should stop trying after a limited number of attempts.

Otherwise, the AP will reply with a `Failure` value containing a human readable message that explains the reason for the failure and then proceed to close the channel.

Transport Protocols

The communication between an agent and AP can take place over any kind of full-duplex binary transport protocol.

In general only minor adaptations are required to make them compatible with CHATS.

TCP (normative)

After opening a TCP connection to the access point, an agent can immediately proceed with CHATS.

The CHATS response returned by the AP will be of type:

```
ChannelSelectionResult (SocketAddress IP4Address)

       or, depending on the kind of IP address of the AP:

ChannelSelectionResult (SocketAddress IP6Address)
```

WebSockets (normative)

In the opening WebSocket handshake, the client sends a single Sec-WebSocket-Protocol header with value `chats`.

The server answers sending a single Sec-WebSocket-Protocol header with the same value.

All further communication takes place using, possibly fragmented, binary messages.

The client then proceeds with the CHATS protocol.

Data Model (normative)

The 正名 types mentioned in this documents (plus all their dependencies) are defined as follows:

```
K3e8257255cbf:
 ADT a b c ≡ ADT {declName :: a,
                 declNumParameters :: Word8,
                 declCons :: Maybe (ConTree b c)}

K07b1b045ac3c:
 ADTRef a ≡   Var Word8
            | Rec
            | Ext a

K4bbd38587b9e:
 AbsRef ≡ AbsRef (SHAKE128_48 (ADT Identifier
                                  Identifier
                                  (ADTRef AbsRef)))

K2e8b4519aeaa:
 Array a ≡   A0
           | A1 a (Array a)
           | A2 a a (Array a)
           | A3 a a a (Array a)
           | A4 a a a a (Array a)
...
                  a
                  a
                  a
                  a
                  (Array a)


Kf139d4751fda:
 BLOB a ≡ BLOB {encoding :: a, content :: Bytes}

K65149ce3b366:
 Bit ≡   V0
       | V1

K306f1981b41c:
 Bool ≡   False
        | True

Kcf6c76b3f808:
 ByPattern a ≡ ByPattern (List (Match (List Bit)))

Kf8844385a443:
 Bytes ≡ Bytes (PreAligned (Array Word8))
```

```
Kc6627a317dbc:
 ChannelSelectionResult a ≡   Success
                            | Failure {reason :: List Char}
                            | RetryAt a

K066db52af145:
 Char ≡ Char Word32

K86653e040025:
 ConTree a b ≡   Con {constrName :: a,
                      constrFields :: Either (List (Type b)) (List (Tuple2 a (Type b)))}
               | ConTree (ConTree a b) (ConTree a b)

K6260e465ae74:
 Either a b ≡   Left a
              | Right b

Kae1dfeece189:
 Filler ≡   FillerBit Filler
          | FillerEnd

K982148c09ddb:
 FlatEncoding ≡ FlatEncoding

K64f93d94a73d:
 HostAddress a ≡   IPAddress a
                 | DNSAddress (List Char)

K0ab5ac6303b9:
 HostPort ≡ HostPort {port :: Word16}

K6cb2ee3ac409:
 IP4Address ≡ IP4Address Word8 Word8 Word8 Word8

K1f3474c12f5d:
 IP6Address ≡ IP6Address Word16
                         Word16
                         Word16
                         Word16
                         Word16
                         Word16
                         Word16
                         Word16

Kdc26e9d90047:
 Identifier ≡   Name UnicodeLetter
```

```
                     (List UnicodeLetterOrNumberOrLine)
              | Symbol (NonEmptyList UnicodeSymbol)

K20ffacc8f8c9:
 LeastSignificantFirst a ≡ LeastSignificantFirst a

Kb8cd13187198:
 List a ≡   Nil
          | Cons a (List a)

Kc23b20389114:
 Match a ≡   MatchValue a
           | MatchAny (Type AbsRef)

Kda6836778fd4:
 Maybe a ≡   Nothing
           | Just a

K74e2b3b89941:
 MostSignificantFirst a ≡ MostSignificantFirst a

Kbf2d1c86eb20:
 NonEmptyList a ≡   Elem a
                  | Cons a (NonEmptyList a)

Kab225802768e:
 PostAligned a ≡ PostAligned {postValue :: a, postFiller :: Filler}

Kb2f28cf37d12:
 PreAligned a ≡ PreAligned {preFiller :: Filler, preValue :: a}

K9f214799149b:
 SHAKE128_48 a ≡ SHAKE128_48 Word8 Word8 Word8 Word8 Word8 Word8

Ke5d02571ce7b:
 SocketAddress a ≡ SocketAddress {socketAddress :: HostAddress a,
                                  socketPort :: HostPort}

Ka5583bf3ad34:
 Tuple2 a b ≡ Tuple2 a b

K7028aa556ebc:
 Type a ≡   TypeCon a
          | TypeApp (Type a) (Type a)

K614edd84c8bd:
```

```
 TypedBLOB ≡ TypedBLOB (Type AbsRef) (BLOB FlatEncoding)

K3878b3580fc5:
 UnicodeLetter ≡ UnicodeLetter Char

K33445520c45a:
 UnicodeLetterOrNumberOrLine ≡ UnicodeLetterOrNumberOrLine Char

K801030ef543c:
 UnicodeSymbol ≡ UnicodeSymbol Char

Kc802c6aae1af:
 WebSocketAddress a ≡ WebSocketAddress {secure :: Bool,
                                        host :: SocketAddress a,
                                        path :: List Char}

Kf92e8339908a:
 Word ≡ Word (LeastSignificantFirst (NonEmptyList (MostSignificantFirst Word7)))

K295e24d62fac:
 Word16 ≡ Word16 Word

K2412799c99f1:
 Word32 ≡ Word32 Word

Kf4c946334a7e:
 Word7 ≡   V0
         | V1
         | V2
         | V3
         | V4
...
         | V123
         | V124
         | V125
         | V126
         | V127


Kb1f46a49c8f8:
 Word8 ≡   V0
         | V1
         | V2
         | V3
         | V4
...
```

```
| V251
| V252
| V253
| V254
| V255
```

References

Flat Serialisation Format (http://quid2.org/docs/Flat.pdf)

TCP (Transmission Control Protocol (https://en.wikipedia.org/wiki/Transmission_Control_Protocol)

WebSocket Protocol (https://tools.ietf.org/html/rfc6455)

正名 (Zhèng Míng) Data Modelling Language (http://quid2.org/docs/ZhengMing.pdf)

About This Document

The only normative parts of this specification are those marked so explicitly *(normative)* , the rest is narrative.

Table 1: Document Metadata.

Status	Early Draft
Version	2017-05-22
First Version	2017-05-22
Reference URL	http://quid2.org/docs/Top.pdf
License	GPLv3
Copyright ©	Pasqualino ”Titto” Assini (tittoassini@gmail.com)