Flight Message Protocols

Version 1.4

November 29, 2011

A. Transport

Hopper's core transport protocol is **AMQP** messaging. We are using the open-source RabbitMQ implementation of AMQP internally. A commercially-supported version of RabbitMQ is available from VMWare SpringSource.

For message serialization, we use **ProtocolBuffers**. You can find out more here: http://code.google.com/apis/protocolbuffers/. It is simple, language-agnostic and much more IO-efficient than XML, which is important considering the message volume that we are dealing with.

B. Messages

For flight data, Hopper currently supports just one type of message: **Trip**, described in detail over the following pages.

C. Consistency

Dropped messages are considered acceptable. They result in corresponding offers being temporarily stale or unavailable in the Hopper search results. In the case of excessive message backlog due to communication failure, it is recommended that messages be dropped. It is recommended that any message buffer have a maximum size of no more than 1 million messages before truncation.

D. Triggers

The Hopper flight protocol is a **Push API**, which implies that the data provider must initiate messages according to its own triggering logic.

The recommended strategy is to trigger a **Trip** message for each of the best available fares returned to a customer request (to the data provider's system) for flights between an Origin and Destination for specific departure and return dates. This strategy ensures that the coverage and freshness of the push stream is consistent with real customer demand.

The following recommendations are designed to limit message traffic to the minimum required by the Hopper search index:

1. It is recommended that data providers trigger no more than 50 messages per client request. In the case of truncation, messages should first be ordered according to a weighted rank of price (75%) and duration (25%).
2. It is recommended that identical client requests received within an hour do not trigger messages. That is to say, for a given Origin-Destination-DepartureDate–ReturnDate request, messages are not triggered if a previous identical request triggered messages less than one hour ago, even if the results are different.

E. Ramp-up

During the early implementation phase of a connection with Hopper, it is recommended that the data provider limit messages as randomly as possible. Some possible approaches:

1. Only send messages from one production instance (depends on provider's architecture)
2. Only send messages for every X request (e.g. modulo on a sequence #).

Trip

The Trip message represents an ordered list of fares that are offered together for purchase by a merchant. For instance, it can represent:

- A one-way trip consisting of one fare
- A return trip consisting of two fares (in the case that the outbound and inbound flights are priced separately)
- A return trip consisting of one fare (in the case that the entire return trip is sold for one price)
- A multi-stop trip consisting of one or many fares

Fieldname	Type	Req	Collection	Description
version	integer			The message version. Defaults to 1
fare	FlightFare	yes	yes	The ordered list of fares, where a fare is a priced unit of travel consisting of one or many segments (see FlightFare class below)
merchant_id	String	yes		The unique ID of the merchant where this trip is offered for sale (4 alpha characters assigned by Hopper)
timestamp	long	yes		The time that this trip was offered for sale. A 64-bit long representing milliseconds since midnight, January 1, 1970 UTC.
booking_path	String			The URL where this trip can be booked. Hopper will refer consumers to this path to purchase the trip.

FlightFare

The FlightFare Class represents an ordered list of flight segments that are priced together under a single fare code and price. For instance, it can represent:

- A one-way direct flight fare consisting of one segment
- A one-way multi-stop flight fare consisting of several segments
- A return trip consisting an outbound flight and an inbound flight that are sold together for one fare code and price only.

Fieldname	Type	Req	Collection	Description
segment	FlightSegment	yes	yes	The ordered list of flight segments included in this fare (see FlightSegment class below).
currency_code	String	yes		The international currency code for the base_amount and tax_amount values.
base_amount	double	yes		The base fare price (not including tax_amount).
tax_amount	double	yes		The additional tax and surcharges that must be added on top the base fare price.
fare_code	String			The fare code, also known as the FBC or Fare Basis Code. e.g. H14ESNR
cabin_class	String			The cabin class, also known as the RBD, or IATA Class Code. See http://en.wikipedia.org/wiki/IATA_class_codes
pax_type	String			The Passenger Type. e.g. CHD for Child.
refundable	boolean			Is this fare refundable? Defaults to false.
fare_rules	String			A free text field allowing human-readable description of the fare restrictions and rules.
source_id	String			The Source of this fare, namely a code representing the GDS or airline where this fare can be booked.
available_seats	integer			The number of available seats remaining for sale at this fare price. Defaults to -1 (unknown).
origin	String	yes		The origin (departure) IATA airport code for the entire fare. The first airport of the first segment of the fare. In the case of a return fare, it also represents the last airport of the last segment.
destination	String	yes		The destination (arrival) IATA airport code for the entire fare. This represents the destination of a one-way or return fare; leave blank in the case of multi-destination fares.

FlightSegment

The FlightSegment class represents a single flight leg by a single aircraft between two airports. Therefore flying from New York to Berlin via London will require two flight segments, one segment from New York to London and another segment from London to Berlin.

Fieldname	Type	Req	Collection	Description
carrier_code	String	yes		The two-character IATA airline code for the carrier of this flight segment.
flight_number	String	yes		The flight number for this flight segment. Usually between 1 and 4 digits. See http://en.wikipedia.org/wiki/Flight_number
origin	String	yes		The origin (departure) IATA airport code for this flight segment.
destination	String	yes		The destination (arrival) IATA airport code for this flight segment.
departure_date	long	yes		The departure date/time of this flight segment. A 64-bit long representing milliseconds since midnight, January 1, 1970 UTC.
arrival_date	long	yes		The arrival date/time of this flight segment. A 64-bit long representing milliseconds since midnight, January 1, 1970 UTC.
stops	integer			The number of stops on this flight segment. Usually 0 unless the airplane makes a stop to refuel. A change in aircraft or flight number indicates a new segment rather than a stop.
outgoing	Boolean			A boolean flag indicating that this segment belongs to the outgoing portion of a roundtrip fare. In the case of rountrip fares only, all segments used to reach the FlightFare destination should be considered outgoing.

FlightProtocols: flights.proto

```
package flights;

option java_package = "com.hopper.models.flights";
option java_outer_classname = "FlightProtocols";

message Trip {
    message FlightFare {
        message FlightSegment {
            required string carrier_code = 1;
            required string flight_number = 2;
            required string origin = 3;
            required string destination = 4;
            required sfixed64 departure_date = 5;
            required sfixed64 arrival_date = 6;
            optional int32 stops = 7 [default = 0];
            optional bool outgoing = 8;
        }
        repeated FlightSegment segment = 1;

        required string currency_code = 2;
        required double base_amount = 3;
        required double tax_amount = 4;
        optional string fare_code = 5;
        optional string cabin_class = 6;
        optional string pax_type = 7;
        optional bool refundable = 8 [default = false];
        optional string fare_rules = 9;

        optional string source_id = 10;
        optional int32 available_seats = 11 [default = -1];

        optional string origin = 12;
        optional string destination = 13;
    }
    repeated FlightFare fare     = 2;

    optional string merchant_id  = 3;
    required sfixed64 timestamp  = 4;
    optional string booking_path = 5;
}
```