Gmail by Google

Qie Hu <qiehu9@gmail.com>

API proposal for conflict resolution system

4 messages

Hao Yi Ong <haoyi@stanford.edu> Mon, Jun 29, 2015 at 6:36 PM
To: Mykel John Kochenderfer <mykel@stanford.edu>, "tomlin@eecs.berkeley.edu" <tomlin@eecs.berkeley.edu>, Qie Hu <qiehu@berkeley.edu>, CASEY DAVID MACKIN <cmackin520@berkeley.edu>, Anayo K Akametalu <kakametalu@berkeley.edu>, Jaime Fernandez-Fisac <jfisac@eecs.berkeley.edu>, Mo Chen <mochen72@berkeley.edu>

Hi everyone,

As promised, I've updated the API proposal and attached it below. Let me know if this makes sense. If there're no issues, I'll send it to Joey and propose a few meeting times for him to choose tomorrow afternoon before I fly to Chicago for ACC. I'll be suggesting anytime on Wednesday or Thursday between 2 pm and 5 pm next week.

Thanks,
Hao Yi

Raw Streaming for UTM Client Server

Functionality

Raw Streaming is an API that gives the conflict resolution system low latency access to the UTM client server's global stream of UAS state information (ID, latitude, longitude, altitude, speed, heading, etc.) within a specific geographic locale. This is similar to how air traffic controllers are assigned to different airspaces for commercial flights. A proper implementation of a Raw Streaming client will be pushed messages indicating the UAS state information and other events have occurred, without any of the overhead associated with polling or worry about API rate limits.

Streaming endpoint

Establishing a connection to the streaming APIs means making a very long-lived HTTP request, and parsing the response incrementally. Conceptually, this is downloading an infinitely long file over HTTP.

Resource URL

tmiserver.arc.nasa.gov/FlightTracker/tracker

Resource information

Information	Description
Response formats	JSON
Requires authentication?	Yes

Rate limited?	No

Example response:

```
{"flightId":"1", "lat":-63.060318, "lon":56.432045, "alt":234.0, "speed":45, "heading":100}
```

Parameters

Parameter	Description
tracks	specifies which aircraft to track with an XML file containing all the UAS IDs; an empty or absence of file automatically subscribes to all active plans

Authentication

The credentials for the UTM system should be no different from the one used for the client credentials, but with an additional flag you can raise with the curl command to switch between client and conflict resolution system interfaces. The default flag should be for the client interface, and the appropriate error response code should be given if access to the conflict resolution system interface is not permitted from the account.

Connecting

To connect to the streaming API, form a HTTP request and consume the resulting stream for as long as is practical. The UTM client server will hold the connection open indefinitely, barring server-side error, excessive client-side lag, network hiccups, routine server maintenance or duplicate logins.

Disconnections

The UTM client server will close a streaming connection for the following reasons:

- A client establishes too many connections with the same credentials. When this occurs, the oldest connection will be terminated. This means you have to be careful not to run two reconnecting clients in parallel with the same credentials, or else they will take turns disconnecting each other.
- A client stops reading data suddenly. If the rate of UAS statuses being read off of the stream drops suddenly, the connection will be closed.
- A client reads data too slowly. Every streaming connection is backed by a queue of messages to be sent to the client. If this queue grows too large over time, the connection will be closed.
- A streaming server is restarted. This is usually related to a code deploy and is not very frequent.
- The UTM client server's network configuration changes. These events are (hopefully) extremely rare, and would represent load balancer restarts or network reconfigurations, for example.

Stalls

Set a timer, either a 90 second TCP level socket timeout, or a 90 second application level timer on the receipt of new data. If 90 seconds pass with no data received, including newlines, disconnect and reconnect immediately according to the backoff strategies in the next section. The streaming API will send a keep-alive newline every 30 seconds to prevent your application from timing out the connection. You should wait at least 3 cycles to prevent spurious reconnects in the event of network congestion, local CPU starvation, local GC pauses, etc.

Reconnecting

Once an established connection drops, attempt to reconnect immediately. If the reconnect fails, slow down your reconnect attempts according to the type of error experienced:

- Back off linearly for TCP/IP level network errors. These problems are generally temporary and tend to clear quickly. Increase the delay in reconnects by 250 ms each attempt, up to 16 seconds.
- Back off exponentially for HTTP errors for which reconnecting would be appropriate. Start with a 5 second wait, doubling each attempt, up to 320 seconds.

Error response codes

Status	Text	Description
200	Success	Self-evident.
401	Unauthorized	HTTP authentication failed due to invalid basic auth credentials, or an invalid auth request.
403	Forbidden	The connecting account is not permitted to access this endpoint.
404	Unknown	There is nothing at this URL, which means the resource does not exist.
406	Not acceptable	At least one request parameter is invalid.
503	Service unavailable	A streaming server is temporarily overloaded. Attempt to make another connection.

Example javascript

```
var websocket;
function streaming_client() {
    websocket = new WebSocket("wss://${username}:${password}@tmiserver.arc.nasa.gov/FlightTracker/tracker");
    websocket.onopen = function(evnt){
        console.log("connected");
        websocket.send("hello");
    };
    websocket.onmessage = function(evnt){
        console.log("received message: " + evnt.data);
        var message = JSON.parse(evnt.data);

        if (message.flighId in tracked_flights){
            if (tracked_flights[message.flighId]){
                flight_track(message.flighId, parseFloat(message.lat), parseFloat(message.lon),
tracked_flights[message.flighId]);
            }
        }
     };
     websocket.onclose = function(evnt){
        console.log("Connection Closed");
        streaming_client();
     };
     websocket.onerror = function(evnt){
        console.log("ERROR: " + evnt.data);
        streaming_client();
     };
}
```

Conflict Advisor for UTM Client Server

Functionality

Conflict Advisor is an API that gives the conflict resolution server the ability to post advisories to the UTM client server, which in turn pushes the advisory to the relevant UAS. Once the advisory is received and acknowledged by the UAS, the aircraft will send a receipt to the client server that also indicates compliance or non-compliance, with a special status code in the case of non-compliance to indicate reason. The resolution server should handle the case of non-compliance based on the reason; e.g., compute a set of new advisories based on reason for all aircraft in potential conflict with non-compliant aircraft, and update these aircraft.

Resource URL

TBD by UTM client server developer

```
tmiserver.arc.nasa.gov/FlightAdvisor/advisor
```

Resource information

Information	Description
Response formats	JSON
Requires authentication?	Yes (resolution server only)
Rate limited?	No

Example response:

```
{"flightID":"10011991", "receipt":"true", "compliance":"true", "reason":"na"}
```

Parameters

Parameter	Description
advisories	specifies which aircraft to send advisories to with an XML file containing all the UAS IDs and the advisory

At present, we assume that all trajectories are in the horizontal plane. Each advisory consists of an ordered sequence of waypoints to start from, accompanied by a set of maneuver parameters that allow the operator to generate the exact trajectory that its aircraft is expected to follow. The trajectory generation will be based on a simple aircraft dynamic model provided separately to all clients, and the waypoint definition adheres to the following schema, which uses SI units for sanity's sake.

Field	Description	Units
`lat`	latitude	NA
`lon`	longitude	NA
`alt`	altitude	m

`speed`	speed in the horizontal plane	m/s
`period`	amount of time to execute maneuver	s
`turn`	turn rate	rad/s

The following is a pair of examples for what might be a set of advisories generated by a conflict resolution server and what the client server will send to a specific aircraft upon receipt of the set of advisories.

Resolution server advisory

```
<wfs:Transaction service="WFS" version="1.0.0"
    xmlns:topp="http://www.openplans.org/topp"
    xmlns:ogc="http://www.opengis.net/ogc" xmlns:wfs="http://www.opengis.net/wfs"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:utm="http://opengeo.org/utm"
    xsi:schemaLocation="http://www.opengis.net/wfs
                        http://schemas.opengis.net/wfs/2.0.0/WFS-transaction.xsd
                        http://opengeo.org/utm">

    <wfs:Insert>
    <utm:advisories>
        <utm:advisory>
            <utm:gufi>fid-15081769</utm:gufi>
            <utm:traj>
                <utm:waypt>
                    <utm:lat>-140.3414</utm:lat>
                    <utm:lon>80.0341</utm:lon>
                    <utm:alt>200</utm:alt>
                    <utm:speed>20</utm:speed>
                    <utm:period>5</utm:period>
                    <utm:turn>-1.3</utm:turn>
                </utm:waypt>
                <utm:waypt>
                    <utm:lat>-143.3414</utm:lat>
                    <utm:lon>81.0341</utm:lon>
                    <utm:alt>200</utm:alt>
                    <utm:speed>15</utm:speed>
                    <utm:period>3.5</utm:period>
                    <utm:turn>0</utm:turn>
                </utm:waypt>
                <utm:waypt>
                    <utm:lat>-145.3414</utm:lat>
                    <utm:lon>83.1249</utm:lon>
                    <utm:alt>200</utm:alt>
                    <utm:speed>20</utm:speed>
                    <utm:period>10</utm:period>
                    <utm:turn>0.42</utm:turn>
                </utm:waypt>
            </utm:traj>
        </utm:advisory>
        <utm:advisory>
            <utm:gufi>fid-01041815</utm:gufi>
            <utm:traj>
                <utm:waypt>
                    <utm:lat>-139.3414</utm:lat>
                    <utm:lon>83.2309</utm:lon>
```

```
                    <utm:alt>200</utm:alt>
                    <utm:speed>20</utm:speed>
                    <utm:period>5</utm:period>
                    <utm:turn>-1.5</utm:turn>
                </utm:waypt>
                <utm:waypt>
                    <utm:lat>-140.8394</utm:lat>
                    <utm:lon>82.3859</utm:lon>
                    <utm:alt>200</utm:alt>
                    <utm:speed>20</utm:speed>
                    <utm:period>9.3</utm:period>
                    <utm:turn>-0.4</utm:turn>
                </utm:waypt>
                <utm:waypt>
                    <utm:lat>-140.3939</utm:lat>
                    <utm:lon>83.0034</utm:lon>
                    <utm:alt>200</utm:alt>
                    <utm:speed>20</utm:speed>
                    <utm:period>4</utm:period>
                    <utm:turn>0</utm:turn>
                </utm:waypt>
            </utm:traj>
        </utm:advisory>
    </utm:advisories>
    </wfs:Insert>
</wfs:Transaction>
```

Client server advisory

The aircraft corresponding to each advisory's flight ID will be sent the advisory that consists of the trajectory information; i.e., the sequence of waypoints and the corresponding maneuver parameters. While the format of the advisory from the UTM client server to the aircraft is to be determined by the developer, we propose the JSON schema:

```
{
    "gufi": "fid-15081769",
    "traj": "waypt": [
        {
            "lat": "-140.3414",
            "lon": "80.0341",
            "alt": "200",
            "speed": "20",
            "period": "5",
            "turn": "-1.3"
        },
        {
            "lat": "-143.3414",
            "lon": "81.0341",
            "alt": "200",
            "speed": "15",
            "period": "3.5",
            "turn": "0"
        },
        {
            "lat": "-145.3414",
            "lon": "83.1249",
            "alt": "200",
            "speed": "20",
            "period": "10",
```

```
            "turn": "0.42"
        }
    ]
}
```

Authentication

The credentials for the UTM system should be no different from the one used for the client credentials, except that the client server recognizes the credentials for the conflict resolution system.

Error response codes

Status	Text	Description
200	Success	Self-evident.
401	Unauthorized	HTTP authentication failed due to invalid basic auth credentials, or an invalid auth request.
403	Forbidden	The connecting account is not permitted to access this endpoint.
404	Unknown	There is nothing at this URL, which means the resource does not exist.
406	Not acceptable	At least one request parameter is invalid.
503	Service unavailable	An advisory server is temporarily overloaded. Attempt to make another connection.

Example call

```
curl -v -u {username}:{password} -XPOST -d @advisories.xml -H "Content-type: application/xml"
"https://tmiserver.arc.nasa.gov/FlightAdvisor/advisor"
```

Qie Hu <qiehu9@gmail.com> Mon, Jun 29, 2015 at 9:19 PM
To: Hao Yi Ong <haoyi@stanford.edu>
Cc: Mykel John Kochenderfer <mykel@stanford.edu>, "tomlin@eecs.berkeley.edu" <tomlin@eecs.berkeley.edu>, Qie Hu <qiehu@berkeley.edu>, CASEY DAVID MACKIN <cmackin520@berkeley.edu>, Anayo K Akametalu <kakametalu@berkeley.edu>, Jaime Fernandez-Fisac <jfisac@eecs.berkeley.edu>, Mo Chen <mochen72@berkeley.edu>

Hi Hao Yi,

Thank you very much for this!
Can you also send us your DASC paper please? We would love to learn more about your work.

Cheers,
Qie
[Quoted text hidden]

Hao Yi Ong <haoyi@stanford.edu> Tue, Jun 30, 2015 at 12:28 AM
To: Qie Hu <qiehu9@gmail.com>
Cc: Mykel John Kochenderfer <mykel@stanford.edu>, "tomlin@eecs.berkeley.edu" <tomlin@eecs.berkeley.edu>, Qie Hu <qiehu@berkeley.edu>, CASEY DAVID MACKIN <cmackin520@berkeley.edu>, Anayo K Akametalu

<kakametalu@berkeley.edu>, Jaime Fernandez-Fisac <jfisac@eecs.berkeley.edu>, Mo Chen <mochen72@berkeley.edu>

Hi Qie,

No problem! I've attached the DASC paper to this email.

Also, as promised during our discussion this afternoon, I've attached a Julia module file that can turn an advisory into a flight path. Specifically, the file has functions that take a well-formed advisory with the sequence of waypoints and maneuver parameters and return a trajectory array with the required speed and heading at each point. I've also attached a Jupyter notebook that contains the exact same code with a simple example of how to use it to visualize the path at the end.

Let me know if you have questions.

Thanks!
Hao Yi Ong

[Quoted text hidden]

3 attachments

root.pdf
246K

TrajectoryGenerator.jl
3K

Trajectory generator.ipynb
41K

Qie Hu <qiehu9@gmail.com> Tue, Jun 30, 2015 at 9:29 AM
To: Hao Yi Ong <haoyi@stanford.edu>
Cc: Mykel John Kochenderfer <mykel@stanford.edu>, "tomlin@eecs.berkeley.edu" <tomlin@eecs.berkeley.edu>, Qie Hu <qiehu@berkeley.edu>, CASEY DAVID MACKIN <cmackin520@berkeley.edu>, Anayo K Akametalu <kakametalu@berkeley.edu>, Jaime Fernandez-Fisac <jfisac@eecs.berkeley.edu>, Mo Chen <mochen72@berkeley.edu>

Thanks Hao Yi!
Qie
[Quoted text hidden]