RIPE Atlas Result Streams

RIPE Atlas has been providing downloadable results from the very beginning of the project. This works well if you know what time frame you're interested in, and want to get to the data collected at that time.

The streaming data service allows you to tap into the real-time data flow of all the collected public results. Every time our system receives a data point or a probe connectivity event occurs, it's also delivered to the clients that are "tuned in" to that result stream. This feature is implemented using web sockets.

Highlights

This service is in *prototype status*. We're observing how our system reacts to the streams provided to users in order to evaluate the feasibility and usefulness of a production service.

You need a websocket capable client and appropriate client side libraries to use it. We recommend using Socket.IO.

You can only subscribe ("tune in") to results delivered by public measurements.

You can subscribe to the connectivity events of any probe.

We're inviting our community to check out the gallery of visualisations provided by the RIPE Atlas team and our users as soon as it becomes available, and to come up with new visualisations and/or to enhance the existing ones.

INTERNAL:
It is also possible to stream historic events at the rate at which they occurred, or with a speedup factor.

More Details

The service address for the stream is http://atlas-stream.ripe.net/stream/socket.io. As explained above, you need to switch to websockets mode.

The snippet below explains shortly how to use our service in a web page.

```
<!-- Load the socket.io client library -->
<script src="http://atlas-stream.ripe.net/socket.io.js"></script>

<!-- Place the JavaScript code below inside <script></script> or in a js file -->
<script>

    // Create a socket and connect to the streaming service
    var socket = io("http://atlas-stream.ripe.net", { path : "/stream/socket.io" });

    socket.on('connect' , function(){ // When the connection is established

        // Set a function to react when an error is received from the channel
        socket.on("atlas_error", callback);

        // Set a function to react when data is received from the channel
        socket.on(CHANNEL_NAME, callback);

        // Subscribe to a channel (you will start receiving data)
        socket.emit("atlas_subscribe", { stream_type: CHANNEL_TYPE, PARAMETERS });

    });

</script>
```

Possible values for stream_type are:

`probestatus`
to receive connection and disconnection events of a probe
the CHANNEL_NAME is "atlas_probestatus"

`result`
to receive measurement results
default if stream_type is missing
the CHANNEL_NAME is "atlas_result"

You can subscribe to multiple streams at the same time, e.g. if you want to visualise the union of results from multiple measurements.

Possible parameters (in addition to stream_type) are:

Name	Description
msm	A specific measurement ID. This parameter is mandatory
prb	A specific probe ID. If you don't set this parameter, you will receive results from all the probes

acceptedFields	A list of accepted fields name, the messages will be pruned server side. If you don't set this parameter you will receive all the fields
enrichProbes	If you want to enrich the information received with the "probestatus" stream about the probes (e.g. lat, long), set this option to true
equalsTo	Allows to filter by values. E.g. with {status: "connected", asn: "3333|4444"} you will receive all the messages with a connected status and ASn equals to 3333 or 4444
lessThan	Allows to filter by values. E.g. with {valueX: 15} you will receive all the messages with a valueX less than 15
greaterTo	Allows to filter by values. E.g. with {valueX: 15} you will receive all the messages with a valueX greater than 15

INTERNAL:
It is also possible (but even more experimental) to play historic events using this API. Note: you cannot have multiple playbacks for the same measurement at the same time (e.g. one year ago and one month ago running concurrently). If you subscribe to a new probe for an already running measurement, you will simply start seeing results for that probe from the already-running measurement stream.

Name	Description
startTime	Start time (seconds since 1970) of the event to replay.
endTime (optional)	End time (seconds since 1970) of the event to replay. For stopped measurements, defaults to the stop time. For other measurements, defaults to the present time.
speed (optional)	The speedup factor (float). Defaults to 1.0 (original speed). There is a limit to the maximum speed which is determined by the number of participants for the given measurement.

Events that you can listen are:

Name	Description
connect	When the connection is established
disconnect	When the connection is terminated
atlas_error	Error messages coming from the stream service
atlas_result	Measurement results of a "result" subscription
atlas_probestatus	Connection and disconnection event of a "probestatus" subscription
atlas_subscribed	When a subscription to a channel is successful
atlas_unsubscribed	When the client has been unsubscribed from a channel

Events that you can emit are:

Name	Description
atlas_subscribe	To subscribe to a channel (parameters needed)
atlas_unsubscribe	To unsubscribe from a channel (parameters needed)

A short practical example:

```
// Subscribe to probe 15 connection events and execute a callback when
// a status is received. The message will contain only the filtered fields
socket.on("atlas_probestatus", function(status){
    console.log("I received " + status);
});
socket.emit("atlas_subscribe", { stream_type: "probestatus", prb: 15, acceptedFields: ["prb_id", "prb_status"] });


// Subscribe to results coming from probe 15 and measurement 1000
// and execute a callback when a result is received
socket.on("atlas_result", function(result){
    console.log("I received " + result);
});
socket.emit("atlas_subscribe", { stream_type: "result", msm: 1000, prb: 15 });
```

```
// Unsubscribe from the last subscription
socket.emit("atlas_unsubscribe", { stream_type: "result", msm: 1000, prb: 15 });

// Do something when the subscription has been terminated
socket.on("atlas_unsubscribed", function(what){
    console.log("The client has been unsubscribed from " + what);
});
```

INTERNAL:

Historical event example

```
// Do something when we get a connection/disconnection
socket.on("atlas_probestatus", function(status){
    console.log("I received " + status);
});

// Do something when the subscription has been terminated
socket.on("atlas_unsubscribed", function(what){
    console.log("The client has been unsubscribed from " + what);
});

// Stream all connection/disconnection events from midnight on 2015-3-18
socket.emit("atlas_subscribe", {
    stream_type: "probestatus",
    startTime: 1426633200
});

// Unsubscribe from the subscription after 5 seconds
setTimeout(function() {
    socket.emit("atlas_unsubscribe", {
        stream_type: "probestatus",
        startTime: 1426633200
    });
}, 5000);
```

You can also check out the source code of the visualisations below.

Available Visualisations

Visualising which DNS root instance a probe ends up on: DNS root instances
Visualising probe connections and disconnections: Probe connections/disconnections