iPlane: Measurements and Query Interface

Harsha V. Madhyastha Thomas Anderson Arvind Krishnamurthy Arun Venkataramani

June 18, 2007

In this document, we describe the application interface to *iPlane*, an information plane that continuously performs measurements to generate and maintain an annotated map of the Internet with a rich set of link and router attributes. As outlined in [1], applications can interface with *iPlane* in the following manner:

- Applications can query and obtain any of the measurements that were collected by *iPlane*.
- *iPlane*'s measurements are limited to those discovered using probes from its vantage points. However, applications might desire path properties between arbitary end-hosts. Hence, *iPlane* exports a query interface through which applications can request *predictions* of path properties between arbitrary end-hosts.
- Applications can choose to upload to *iPlane* a few (typically 10) traceroutes outgoing from their clients. The traceroutes from a client help *iPlane* capture the routing diversity near that client for better predicting the performance of paths from it. While this information improves prediction accuracy, *iPlane* can also operate without having this information.

iPlane maintains a database-like view of path properties between pairs of end-hosts. For every source-destination pair, there exists a row in the view with *iPlane*'s predicted path between these hosts and the predicted performance of the path. Any query to *iPlane* involves an SQL-like query on this view – selecting some rows and columns, joining the view with itself, sorting rows based on values in certain columns, and so on.

We provide tools that applications can use for interacting with *iPlane*. We also make the source code for all our tools publicly available. In subsequent sections, we describe the tools we provide for uploading traceroutes and for issuing queries to *iPlane*.

1 Uploading Traceroutes

We provide a Java class called *iPlaneTracerouteUploader*, which can be executed on clients to perform a small number of traceroute probes and upload the measurements to the *iPlane* server.

The usage of this class file is as follows:

```
java iPlaneTracerouteUploader <server> [--trc_cmd <traceroute_command>] [dest_1] ... [dest_n]
```

iPlaneTracerouteUploader determines the operating system running on the client on which it is executed. It uses this information to determine the appropriate command to use for issuing traceroutes—`tracert` on Windows, and `traceroute` on Linux and Mac OS X. Users can override the default traceroute command by specifying an alternate one with the `trc_cmd` command-line argument. The set of destinations to which traceroutes should be issued can also be provided as command-line arguments. If no destinations are provided as arguments, *iPlaneTracerouteUploader* issues a HTTP GET request to a standard webserver. This webserver, currently hosted on *iplane.cs.washington.edu*, returns a set of 10 destinations in randomly chosen BGP prefixes. Traceroutes are then issued; each traceroute is terminated once three successive unresponsive hops are encountered. Finally, the output from these traceroutes is uploaded to a *iPlane* server, specified by the `server` argument, on port 7820. An *iPlane* server is currently running on *iplane.cs.washington.edu*.

2 Issuing Queries

Applications can query *iPlane* for path properties between arbitrary end-hosts. If the requested path has been measured by *iPlane*, it returns the measured properties for the path, else it uses its available measurements to predict the path properties. Every *iPlane* server exports an SQL-like view of its predictions by means of a SUN RPC and XMLRPC interfaces.

2.1 SUN RPC interface

Client applications can issue queries on the exported view of performance predictions by uploading Ruby scripts through an RPC interface. This can be done using the tool *query_iplane_client* described later. The uploaded script can issue multiple queries to *iPlane*, thus providing performance improvements when compared to the direct approach of issuing each query as a separate RPC.

The uploaded Ruby scripts should contain the statement `require 'iplane'` to link with *iPlane*'s library. The library implements two classes—*IPlane* and *IPlaneResponse*. The *IPlane* class exports the following methods in order to enable batched execution of queries:

- `addPath(src, dst)`
  IPlane maintains a queue of paths for which predictions need to be made. A call to `addPath` adds the path from *src* to *dst* to this queue and immediately returns. *src* and *dst* need to be IP addresses; DNS names do not work in the current implementation.

- `queryPendingPaths`
  When a call to `queryPendingPaths` is made, *iPlane*'s prediction engine is invoked to predict the path properties along the set of paths currently in *IPlane*'s queue. This call blocks until predictions for all enqueued paths have been determined. An array of type *IPlaneResponse* is returned, with one element in the array for each predicted path.

The member variables of the class *IPlaneResponse* are *src*, *dst*, *path*, *cluster_path*, *as_path*, *meas_or_pred*, *latency* and *loss*. These correspond to the source and destination, arrays containing the router interface level hops, cluster level hops, and AS level hops along the path between these hosts, a flag that is set to false if this path exists in the measured atlas or to true if the path had to be predicted, and the end-to-end latency and loss rate on this path.

Below is an example script that takes as input a client and a set of replicas. The script determines the replica closest, in terms of latency, to the client.

```
#!/usr/bin/ruby

require 'iplane'

client = ARGV[0]
replicas = Array.new
(1..ARGV.length-1).each { |i|
    replicas.push(ARGV[i])
}

iplane = IPlane.new
replicas.each{ |r|
    iplane.addPath(client, r)
}
responses = iplane.queryPendingPaths

min_lat = 3000 #3 seconds
best_replica = ""
responses.each{ |r|
    if (r.latency < min_lat)
        min_lat = r.latency
        best_replica = r.dst
    end
}
puts "#{best_replica}"
```

Queries are issued via an RPC interface. The server end of this interface currently runs on *iplane.cs.washington.edu*. We provide a tool called *query_iplane_client* that applications can use to issue queries [1]. Versions of *query_iplane_client* that run on Linux, Mac OS X, and Windows are provided. The usage of this tool is as follows:

[1] An as yet unimplemented feature is that clients can issue queries to a standard name, which resolves to the PlanetLab node closest to the client.

```
./query_iplane_client <server_host> <script_invoke_type> <script_filename> [argument_list]
```

The `server_host` command-line argument is the hostname or IP address of the node to which the query is to be issued (currently, *iplane.cs.washington.edu*). The `script_invoke_type` argument specifies how the script that executes the query is provided as input. A value of 1 corresponds to a call-by-value invocation; the script is read in from the filename specified by the `script_filename` argument. In addition, *iPlane* hosts a standard set of scripts (whose filenames will be published), for scenarios such as predicting the path and path properties between a source-destination pair, predicting the best set of peers for each host among a set of hosts, predicting the best replica to serve a given client among a set of hosts, and so on. While using such predefined scripts, the `script_filename` argument can also refer to any of these scripts by name, by using a value of 2 for the `script_invoke_type` argument. `argument_list` is an optional string argument that will be provided as the command-line argument to the script, when executed by *iPlane*. Multiple arguments can be provided to the script by making `argument_list` a space-separated concatenation of all the arguments enclosed within single-quotes.

2.2 XMLRPC interface

iPlane's query server also exports an XMLRPC interface, to which clients can issue queries. The XMLRPC interface exports a single method *iplane.query*. To enable batching of queries, this method accepts as its argument an array of paths. Each path is represented by a structure with two named members *src* and *dst*, which map to the source and destination IP addresses in string format. The *iplane.query* method returns an array of structures as its response, where each structure in the array contains *iPlane*'s predictions for one of the paths in the input argument. Each structure in the returned array has the following named members:

- *src* and *dst* are strings that map to the source and destination IP addresses.
- *as_path* and *cluster_path* are arrays of integers containing the AS level and cluster level hops on the path.
- *path* is an array of strings containing IP addresses of the router interface level hops on the path.
- *meas_or_pred* is an integer set to 0 if this path exists in the measured atlas, or to 1 if the path had to be predicted.
- *latency* is an integer value set to the end-to-end latency.
- *loss* is a double value set to the end-to-end loss rate.

The server end of this interface currently runs on *tito.cs.washington.edu*. Further details on how to issue queries to the XMLRPC interface are available upon request.

References

[1] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *Proc. of the Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.