Multipath Service OSCARS Client

The Multipath service client for OSCARSv0.6 is intended to be used as a project independent from OSCARS, and to be run from machines with or without an active OSCARS instance running. It has been designed to make full use of the provided OSCARS API framework, while giving the end-user an almost identical experience in terms of ways to create/query/cancel reservations. The Multipath client allows for the manipulation of virtual circuits (VCs) in a collective manner. A user may therefore request a new reservation between a specific source and destination in the topology. Through operations transparent to the user, a collection of point-to-point (unicast or anycast) circuits will be set up entirely by OSCARS and combined under the umbrella of a group global reservation identifier (MP-GRI). The intention is that if the user indicates that two paths are desired for a single Multipath reservation, each of its unicast sub-requests will be link-disjoint from all other sub-requests within that same Multipath group. Meanwhile, the core behavior of OSCARS is completely unaltered, so each of the sub-requests in the group still has its own unicast GRI. In this way, requests may be manipulated as a group, or individually, with no effect on the user experience.

PROJECT STRUCTURE

The source code of the Multipath service is a (mostly) front-end client application written in Java, so it is designed to be platform independent, and does not alter the behavior of OSCARS but is able to communicate directly with its services. Emphasis was placed on keeping the user experience as similar to that of the OSCARS API as possible. As long as the user has access to a running instance of OSCARS (either on the same machine or on a remote machine such that the user knows the IP address) he will be able to use this client. The overall structure of the project is as follows

The project folder is named `AnycastMultipathClient`.

The project folder includes several subdirectories:

- `src` contains all the Java code used to implement the Multipath client, including the code used to implement the graphical user interface (GUI), as well as the code that parses input from the command line when various scripts are executed.
- `lib` contains dependencies and jar files needed for the Multipath client to operate successfully and maintain contact with OSCARS.
- `cli` houses bash command line interface (CLI) scripts used to create/query/cancel, reservations, list reservations by status or by group, and combine existing reservations into new or existing groups. Those familiar with the OSCARS API should recognize that these scripts expect the same input as those in the $OSCARS_DIST/api/bin folder.
- `yaml` contains several sample parameter files for use with the scripts in cli when submitting a request for a new reservation. Note: YAML files are sensitive to white-space.
- `images` holds a set of images used for the Demo Graphical User Interface.

The `src` folder is further decomposed into several packages:

- `multipath` contains the core functionality of the Multipath client, including the primary class: `MultipathOSCARSClient`. This class provides the fundamental behavior necessary for group-based operations, as well as maintaining functionality for performing operations on individual reservations in an identical manner, completely transparent to the user. This package also holds several helper classes which exist to keep `MultipathOSCARSClient` clean and straightforward. These helpers are mostly essential to the behavior of the core class.
- `cli` contains a single class with a main method whose mission is to interpret input from the CLI scripts, package it into the appropriate format, and forward it to `MultipathOSCARSClient`.
- `gui` contains a view/model and controller class which provide for a fully-functional graphical user interface (GUI) with which users may submit and handle/monitor both Multipath and Unicast reservations. This GUI is designed mostly for demonstration purposes, but can be used as a direct interface to the system if desired (single-domain reservations only).
- `demoGui` holds a set of classes which provide a streamlined GUI for demonstration purposes. This GUI allows a user to track current reservations in OSCARS, and place unicast reservations with a specified source, destination, and number of paths. Other parameters available in the interface located in `gui` are set to default values to simplify the reservation process.
- `test` contains two classes which are designed to provide sample scenarios for interacting directly with the `MultipathOSCARSClient` interface and give a clear understanding of the output that results from various input scenarios for its core methods.
- `conf` contains one class with parameters necessary for setting up and handling WebServices security parameters, which must necessarily be handled in order to communicate with OSCARS. The contents of this class must be altered by the user when setting up a new instance of the Multipath client.

GETTING STARTED

The client has been designed to be easily compiled and started on any platform regardless of whether the CLI or GUI interface will be used. If the preferred method is to use the GUI, it is recommended that you import `MultipathClient` as a java project into an IDE (such as Eclipse).

Modifying OSCARS

If the Multipath client project is being run on a machine containing a running instance of OSCARS, no alterations are necessary to OSCARS itself. If however, OSCARS will be run on a separate machine, you will need to alter the `TopoBridge` WS to be deployed on a specific IP address rather than localhost, which is the default setting. In `$OSCARS_HOME/TopoBridgeService/conf/`, you will need to modify the `config.HTTP.yaml` and `config.SSL.yaml` files to publish to the IP address of the machine on which OSCARS is running. Then you can restart the `TopoBridge`, and you should be all set:

```
$ cd $OSCARS_DIST/
$ bin/stopServers.sh topoBridge
$ bin/startServers.sh dev topoBridge
```

(To start in development mode).

The Multipath client is *mostly* a front-end application which performs its own tasks through direct communication with the OSCARS API. However, in order to create link-disjoint paths, OSCARS itself needs to be altered. Specifically, the Path Computation Engine (PCE) responsible for computing shortest paths, **DijkstraPCE**, must be replaced by the custom version included in the root directory of the Multipath client project. The file to replace can be found in **$OSCARS_DIST/dijkstraPCE/src/main/java/net/es/Oscars/pce/dijkstra/DijkstraPCE.java**
Once this file has been replaced, OSCARS will need to be recompiled and the servers started.

Setting parameters
Now you must set the WS-Security parameters needed to establish communication with OSCARS. These parameters can be found in **MultipathClient/src/conf/Configuration.java**.
You will need to set *oscarsURL* to the URL and port on which your instance of OSCARS is running.
You will need to set *topoBridgeURL* to the URL and port on which your instance of the **TopoBridge** WS is running. Usually the OSCARS port is ***:9001***, while the topology bridge port is ***:9019***. You should also set *topologyDomain* to the local topology set in OSCARS. This information ensures that the GUI source/destination lists display properly.

You must then also set the certificate keystore for establishing a certified connection with OSCARS. If you are deploying the Multipath client on the same host where OSCARS is running, you can set the keystore information as follows:

```
keystoreClient = "$OSCARS_HOME/sampleDomain/certs/client.jks";
keystoreClientUser = "mykey";
keystoreClientPasswd = "changeit";

keystoreSSL = "$OSCARS_HOME/sampleDomain/certs/client.jks";;
keystoreSSLPasswd   = "changeit";
```

If you wish to run on a separate host, you will need to provide the locations of your keystores. This guide does not cover setting up your own keystores, but information can be found on the OSCARS Wiki.
That's it for set-up, you are ready to go!

Compiling

With an IDE, you don't have to do anything to compile. You can just start up one of the main classes and begin using the Multipath client. Output will be printed to the console reflecting what is happening in the client.

Alternatively, a compile bash script is provided for Linux systems. From the project's root directory, simply run:

```
$ ./compile.sh
```

This not only builds the source code, but also sets all the scripts to be executable. If desired, you can also launch the GUI form the command line after compiling by executing the command:

```
$ ./runGui.sh
```

To run some of the CLI scripts, you can try:

```
$ cli/createRes.sh
$ cli/queryRes.sh
$ cli/cancelRes.sh
```

(The above three commands do not show their respective arguments, but that information will be displayed by the called main() method of the source code in the `cli` package.)

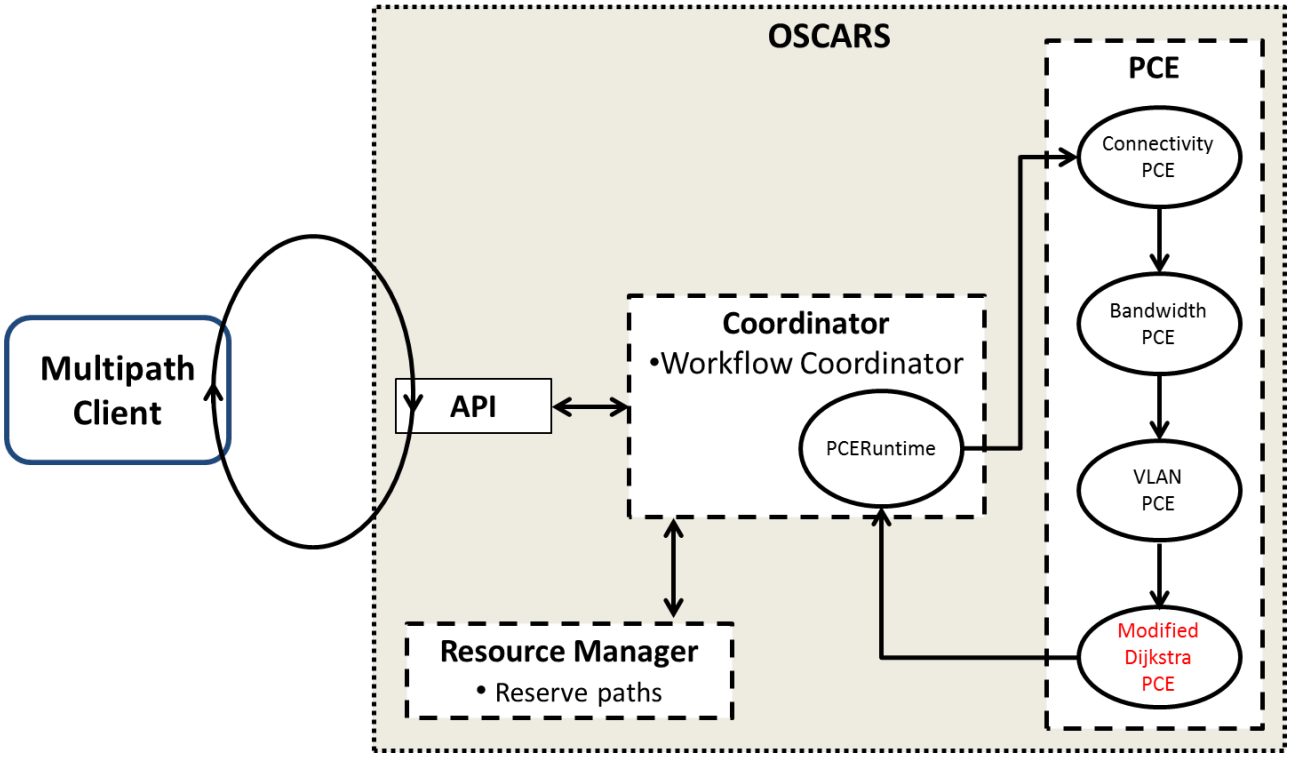
*At this point you should have a fully-functioning Multipath client deployment. NOTE: The `MultipathClient` project was developed for Java 7. If you are using Java 6, you may have to remove the parameterized type `<Object>` from several of the swing components in the `/src/gui/` source files. Removing the parameterized types will have no adverse effect on the behavior of the GUI or any other portions of the Multipath client.

HOW THE CLIENT WORKS

The client bundles up information from the user into formats usable by OSCARS. The OSCARS API is very straightforward and usually follows this format:

So for example, in submitting a request for a new reservation, the client will first take in the user's parameters, and package it into a `CreateResContent` object, which will be passed to the OSCARS API. OSCARS will respond with a `CreateReply` object containing the GRI of the new reservation and

its status in the system. By handling several of these request/response objects together (to simplify, think of it as a loop of individual interactions with OSCARS), sub-requests may be logically combined into Multipath groups. For instance, if a user submits a request for a new reservation with two link-disjoint paths, OSCARS will establish two GRIs, one for each sub-request, and the Multipath client will assign an MP-GRI representing BOTH of those individual GRIs. The user may then use this MP-GRI to perform other group operations on multiple sub-requests together. For example, if the user wants to get the status of all members of a group, he simply has to pass in the MP-GRI for that group, and all the statuses will be returned and printed for EACH sub-request. Similarly, if the user wishes to cancel multiple reservations together, he may pass in the MP-GRI to Multipath client's `cancelMPReservations()` method and all sub-requests will be cancelled together.



The above figure gives a more descriptive view of the Multipath client's interaction with OSCARS when creating a group reservation. One iteration through the loop represents the result of invoking a `createReservation()` call in OSCARS. The Multipath client interacts directly with the OSCARS API, which forwards the request to the Coordinator. The Coordinator then invokes a call to the PCE stack, which includes the modified `DijkstraPCE` detailed previously. The result of one execution through this stack is a VC or a FAILED reservation. The details of the request are then passed back through the OSCARS API to the Multipath client. Upon receiving a reply from OSCARS, the Multipath client polls the reservation to determine if the reservation was successful or unsuccessful. In the case of an unsuccessful reservation the Multipath client stops trying to create disjoint paths and exits. If the reservation is successful, then a second iteration of the loop occurs, following the same trajectory as the first with the exception that the new call to `createReservation()` passes in a request with the

first path embedded into the new reservation request's *OptionalConstraints*. The modified `DijkstraPCE` uses these *OptionalConstraints* to ensure that the second path is link-disjoint from the first. This looping continues until network resources prohibit the construction of a link-disjoint path or the user-specified number of disjoint paths have been successfully established.

More about grouping

`MultipathClient` was developed with a strong focus on the grouping functionality. For this reason it is quite flexible. The user doesn't necessarily have to submit a Multipath request to create a new reservation with multiple link-disjoint paths in order to handle a group. The user also has the freedom to add new paths to *existing* reservations. This allows a Multipath reservation with two link-disjoint paths to become a Multipath reservation with (for example) three link-disjoint paths. The new path added will be link-disjoint from the pre-existing group members. Note: If the last-created sub-request does not have a status of ACTIVE or RESERVED, a new link-disjoint member will NOT be added to the group as its establishment would be superfluous. Also note that if previous members (with the exception of the last-created member) of a Multipath group are CANCELED/FAILED/FINISHED, a new member will still be link-disjoint from ALL other group members, including the CANCELED/FAILED/FINISHED members, despite the fact that their paths are no longer reserved. It is possible to add multiple new paths to an existing group, but the GUI and code in the `/src/test/` folder provide examples of adding a single new path at a time. A pre-existing unicast GRI may be "duplicated" with a new link-disjoint partner, as long as the original unicast GRI is ACTIVE or RESERVED. If the original unicast GRI is already part of a group, then the new path will be added to that group. Otherwise, a new MP-GRI will be established to include both the original and new link-disjoint GRIs.

Items may also be removed from groups once they have been added. In the current design **EVERY GROUP MUST BEGIN WITH THE SUBSTRING "MP"**. There can also not be any duplicate MP-GRIs. Attempting to create a duplicate MP-GRI will simply result in appending to the original group with that MP-GRI. These MP-GRIs are stored in a lookup file, named ***"MP_lookup_gri.txt"*** by default. It is not recommended that you directly alter this file, as it will affect your group structures. Note however, that deleting groups from this file DOES NOT cancel or destroy the unicast sub-requests therein as those are maintained entirely by OSCARS.

CREATING MULTIPATH RESERVATIONS

As previously mentioned, the term Multipath is used as a general term for reservations with multiple link-disjoint paths between the same source and destination pair. The behavior of `MultipathClient` is modified simply by changing the input parameter to a new reservation request. Each request for a new reservation contains one additional parameter to the traditional OSCARS API. This parameter specifies the number of intended link-disjoint paths for a single Multipath reservation group. For example, by passing in an argument of "2", a link-disjoint pair of paths will be established (assuming available resources). Such a scenario is useful for providing a survivable hot-backup path in the case of a primary path failure. The Multipath client will perform best-effort VC establishment. In other words, if

the user (or third-party client application) requests a Multipath request with *M* disjoint sub-reservations, and the network topology only has enough resources to support *N* disjoint paths, such that *N* < *M*, then only *N* paths will be reserved. The Multipath group reservation will thus contain (*N*+1) sub-requests, with the last entry being a sub-request with status: FAILED.

All members of a Multipath reservation are homogenously handled. This means that if *M* link-disjoint paths are requested, and assuming network resources allow for all *M* paths to be established, then all *M* of the sub-requests will have the same source URN, destination URN, description, start-time, end-time, etc. It is possible to modify (some of) these parameters individually by passing the sub-request GRI to the Multipath client's `modifyMPReservation()` method. In this way, a homogenous Multipath group can become a group of heterogeneous sub-requests. Also note: when adding a new sub-request to an existing Multipath group, the parameters of the most-recently created member of that group will be "cloned." In this manner, even if members of the group have been modified, the new member will take on the same behavior as that currently specified by the most recently created member of the group.

ANYCAST REQUESTS

The Multipath Client has been updated to include Anycast functionality. Anycast, as opposed to Unicast, allows for the selection of any one out of a set of destinations for a reservation, with the selection based on the current network state. OSCARS can be made Anycast-capable by replacing the content of your OSCARS installation's *DijkstraPCE* with the content from this project directory's *DIjkstraPCE_withAnycast*.java file. With this change (and recompilation of your OSCARS code), Anycast requests can be made both directly through OSCARS, and through the Multipath client.

ANYCAST DESTINATION FORMAT

A typical Destination URN for a Unicast request appears as follows:

urn:ogf:network:domain=es.net:node=DENV:port=port-4:link=link1

The key differences between unicast and anycast come in with the "node=_____" and "port=_____" substrings. The node field in the anycast URN contains an indicator that the request is anycast, and lists out the destination node names and the corresponding port numbers. The format is as follows:

"node=anycast(*NodeName1-PortNumber,NodeName2-PortNumber...)"*.

The port field in the URN changes simply to **"port=*anycast*"***.* This indicates to OSCARS that it

should look for a port number in the anycast destination set that will correspond to a particular node. All together, a destination URN appropriate for the anycast version of OSCARS would appear as:

urn:ogf:network:domain=es.net:node=anycast(DENV-4,SUNN-5):port=anycast:link=link1

ANYCAST MODIFICATIONS TO MULTIPATH CLIENT

A class, *AnycastHandler.java*, was added, and small modifications were made to the *MultipathOSCARSClient.java*, *GUIController.java, MultipathUI.java,* and *MultipathCLIMain.java*, to enable Anycast request handling. Anycast request handling comes in two flavors: **Anycast Before Multipath (ABM)**, or **Anycast Multipath with Minimum Hops (AMMH)**, **which** can be switched between by changing the value of the variable `boolean smartToggle` in *AnycastHandler.java*. The two favlors determine how a request is satisfied – either more efficiently, but not necessarily with the lowest cost (in total hops), or with a longer running time and the ideal solution.

ABM

An anycast request is passed into the Multipath client through one of the above outlined methods (CLI or GUI), and is sent into the OSCARS API, as with unicast requests. A reply is returned, and the Multipath client determines which destination was chosen out of the Anycast destination set. It then sends in all remaining requests for disjoint paths (if any) as Unicast requests to that destination, mimicking the regular process performed for Unicast Multipath requests. While this will successfully create survivable disjoint paths to the destination chosen as “the best” (minimum hops for the first path) by the Anycast distribution of OSCARS, the total number of hops across all disjoint paths in this Multipath reservation may not be minimum. In order to determine which Anycast destination in an Anycast-Multipath request will result in the minimum number of hops total, **AMMH** can be activated.

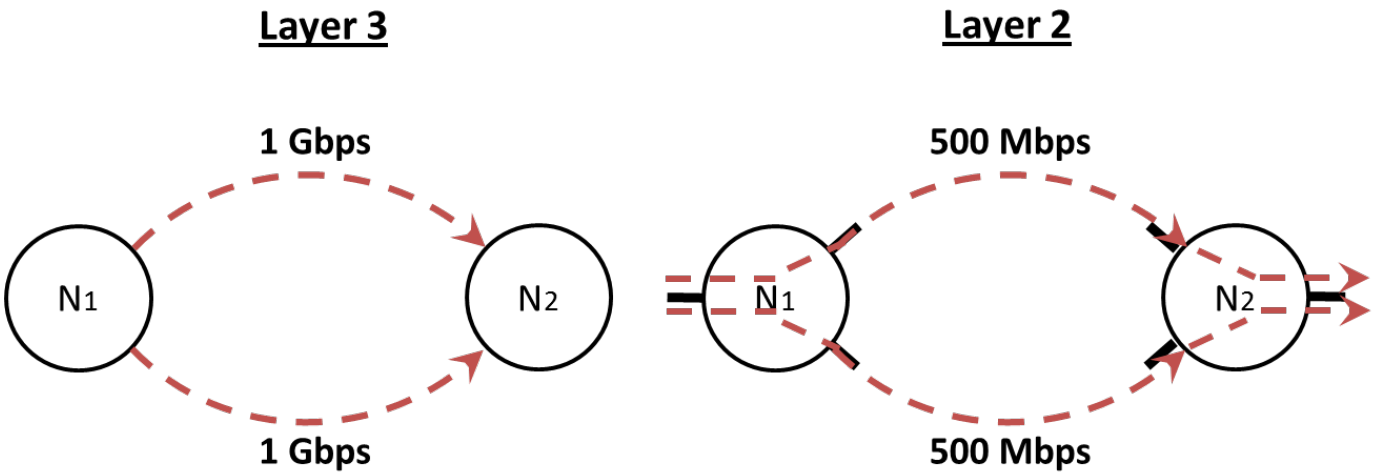
AMMH

Functionally, **AMMH** is comprised almost entirely of performing the actions in **ABM** repeatedly, once for each destination in the destination set. An Anycast request is passed in, the required subrequests using disjoint paths are reserved, metrics on the Multipath reservation are recorded, and then the reservation is cancelled. The chosen destination is removed from the specified destination set, and a new Anycast request is passed to the OSCARS API with an updated, smaller destination set. This process is repeated until all Anycast destinations in the set have had Multipath reservations made, and then the metrics for each destination’s requests are compared. The request that resulted in the minimum number of hops across all routed

paths in the Multipath reservation is selected as the best, and a Unicast Multipath reservation is made for this destination. If best effort subrequest reservation is desired, (determined by the value of the `boolean flexible` variable in *AnycastHandler.java*) the destinations are also compared by prioritizing the Multipath reservation that resulted in the maximum number of successful paths, with ties broken by minimum hop count.

DESIGN LIMITATIONS

The current implementation of the Multipath client is not without its shortcomings. Chief among them is the fact that OSCARS currently only supports layer-2 reservations. This means that each source/destination URN consists not just of a network node, but a port/link connected to that node. When viewed from a layer-3 perspective, having multiple paths between a source and destination gives the impression that bandwidth capacity of a reservation is increased. However, the layer-2 support creates a bottleneck at the source and destination ports. The figure below provides an illustration of this bottleneck. Consider a VC between nodes N_1 and N_2 with 1 Gbps available bandwidth along each path. Since the ports at N_1 and N_2 are shared between the link-disjoint paths, the maximum bandwidth that can be established along each path is only 500 Mbps, or half the total link-capacity.



Another design limitation derives from the fact that a Multipath reservation consists of a number of unicast sub-requests. Each of these sub-requests has its own GRI and therefore, each is established on a separate VLAN. This requires the Multipath client to provide a VLAN lookup table to map to sub-requests. Relying on a lookup table ultimately lowers end-user transparency, as the user loses the illusion that the Multipath reservation is a single entity. This design limitation may be avoidable with future releases of OSCARS that may support grouping multiple paths under a single OSCARS GRI

MISCELLANEOUS NOTES

Refer to the in-code documentation for clarification on what parameters are expected for each operation.

When testing the YAML files from the CLI scripts, you may have to update some of the start/end timestamps as they might be in the past as of your attempt to use the system. The format is always the same in these files: ***"YYYY:MM:DD HH:mm"***.

The test-cases written in the classes in `src/test/` assume the user has the "es.net" topology as the local-domain.

This project was developed solely by members of the Advanced Computer Networks Laboratory group at the University of Massachusetts, Dartmouth, and later at the University of Massachusetts, Lowell. Any questions can be addressed to the student developers Jeremy Plante and Dylan Davis, or to the project advisor Dr. Vinod Vokkarane:

jeremy_plante@student.uml.edu
Dylan_davis@student.uml.edu
Vinod_vokkarane@uml.edu