Appendix C
Quantum API and Extensions for Simulating Network Resource Reservation

The following proposes how one might use the Quantum API to simulate the network controller API operation that is described in Appendix B. OpenStack Quantum is a system for managing network IP addresses for cloud systems. Quantum is capable of setting up a virtual network infrastructure for cloud computing and therefore, it can be useful in this project for simulating a network infrastructure.

In this appendix, I go over the important points of the Quantum API as it applies to this project. In addition, I propose a few parameter extensions for existing Quantum resources as well as a resource extension. The full API documentation and other Quantum information can be found at https://wiki.openstack.org/wiki/Quantum.

Like the “real” API described in Appendix B, Quantum is a RESTful API. I adopted many conventions of the Quantum API along with other APIs in Appendix B, so there are many similarities.

Quantum Resources:
Network
Subnet
Port
EXTENSION - Route

For simulating P2P network resource reservation, I propose using Networks resources to simulate the overall topology and Subnets to specify IP ranges of devices in individual P2P sessions. Port resources can be used to define virtual “devices” on the network

Basic Quantum API format:
GET /<objects> - List summary of all objects of type <objects>
GET /<objects>/<object-id> - List detailed info for specific object
POST /<objects> - Create new object of type <object>. Requires JSON payload
PUT /<objects>/<object-id> - update object with id. JSON payload of attributes to change and new values
DELETE /<objects>/<object-id> - delete object with id

This API format is used in approximately the same way as the API described in Appendix B. For details on the individual requests and JSON payload formats, see the Quantum API documentation online. Anything marked “EXTENSION” is not in the original Quantum API.

Responses are JSON payloads that represent the complete details of the resource requested or created.

Creating network:
   POST /networks
       name
       admin_state_up (basically turn off the network)
       EXTENSION app_name (API stores the name of the application in use)
   Updating: PUT /networks/network-id with same parameters

Creating subnet:
   POST /subnets
       network_id
       ip_version
cidr
allocation pools (for IP ranges)
EXTENSION bandwidth
EXTENSION session_name
EXTENSION peer_count
EXTENSION peers_list
EXTENSION session_id
EXTENSION routes
List of routes. Added by creating routes with a subnet_id (see below)
Update with PUT /subnets/subnet-id with same parameters.

Creating Port:
POST /ports
name
device_id
network_id
fixed_ips
only include subnet_id in this
EXTENSION peer_info
  EXTENSION peer_id   (original ID in the P2P network
  EXTENSION peer_ip   (original IP in the P2P network)
Update with PUT /subnets/subnet-id with same parameters.

The Route resource is a proposed extension to the quantum API for resource reservation
Create Route
POST /routes   - (Same API as other objects. See above)
source_peer   (origin peer ID for hose-model bandwidth reservation)
dest_peers    (list of destination peer IDs for hose-model bandwidth reservation)
capacity      (capacity to guarantee)
subnet_id     (ID of subnet resource that this route applies to)
Update with PUT /routes/route-id

Key points:
• Burden of determining routes on tracker rather than on network controller
  o Same number of commands (O(n^2)) but they are done during the network
    construction phase rather than afterwards all at once
• Uses the basic Quantum API structure and adapts the API resources already in place with
  minor additions
• Batch updates? They work but it’s not necessarily good practice to do the whole network at
  once because the complexity of the network graph is O(n^2)

Security Considerations
• API commands should be sent via HTTPS
• Quantum uses the keystone identification service for security
  o Keystone provides both authentication and authorization
• Logins and cookies violate the RESTful principles which are important for minimizing state
  load
• We could use OAuth to protect the API against use by unauthorized parties
  o Tracker would present OAuth tokens on API calls
• Another possibility is query signing, wherein the tracker would sign each RESTful request