# Appendix C

# Network API

We present a sampling of the network-API we used in the prototypes discussed in Chapters 2, 3 and 5.

```
// Create a virtual port as an interface between packet and circuit
// switching domains
void createVirtualPort(uint64_t switchId, uint16_t vportId);

// Update virtual port with circuit-flows defined by ofp_tdm_port.
// @insert  true adds circuits to the vport, false deletes them.
// @cflowId caller-defined ID to identify circuits
void updateVcgWithCflow(uint64_t switchId, uint16_t vportId,
                        struct ofp_tdm_port& tpt, bool insert, uint32_t cflowId );

// Update virtual port with packet-flows defined by ofp_match
// Currently bidirectional packet flows are created such that:
//  -- vlan tags are added to packet that match before being forwarded
//     out of the virtual port
//  -- van tags are matched in the reverse direction along with virtual-ports as
//     in_port
// @insert  true adds permanent packet-flows to the vport, false deletes them.
// @cflowId caller-defined cookie to identify permanent flows
void updateVcgWithPflow(uint64_t switchId, uint16_t vportId, ofp_match pflow,
                         uint16_t vlanid, bool insert, uint64_t cookie );

// Poll port stats for virtual port
void pollPortStatsReq(uint64_t switchId, uint16_t vportId);

// Increase bandwidth along the path defined in Route.
// @ addBw bandwidth to add in Mbps. If bandwidth is quantized (eg. time-slots)
//   the closest value not greater than addBw will be used
void increaseBandwidth(uint32_t addBw, Route& path);

// Increase bandwidth along the path defined in Route.
void decreaseBandwidth(uint32_t delBw, Route& path);
```

```
// Create a bidirectional cross-connection defined in tpt1 and tpt2
// as part of the circuit flow identified  by cflowId
void addXconn(uint64_t switchId, struct ofp_tdm_port& tpt1,
              struct ofp_tdm_port& tpt, uint32_t cflowId );

// Delete the bidirectional cross-connection identified by cflowId
void deleteXconn(uint64_t switchId, uint32_t cflowId );

// Setup a 'flow' along a non-MPLS_tunnel 'route' - thus the definition
// of a flow is the same at each switch along the route. The only action
// applied is OFPAT_OUTPUT. Everything in Flow should be in network byte order.
// @nw_inport and @nw_outport refer to the ports where packet enters and exits
// the network, and should be specified in host byte order
bool setupFlowInIPRoute(const Flow& flow, const Routing_module::Route& route,
                        uint16_t nw_inport, uint16_t nw_outport,
                        uint16_t flow_timeout );

// Setup a 'flow' along a 'route' which includes an MPLS tunnel.
// Instead of the information passed in @'route', we use
//   -- @srcBr (tunnel-head) label information @srcOutLabel is pushed and outport
//      @srcOp is used
//   -- @dstBr (tunnel-tail) the flow definition changes to the physical in-port
//      @dstInPort
// Everything in Flow should be in network byte order.
// nw_inport and nw_outport refer to the ports where packet enters and exits the
// network, and should be specified in host byte order
bool setupFlowInTunnelRoute(const Flow& flow, const Routing_module::Route& route,
                            uint16_t nw_inport, uint16_t nw_outport,
                            uint16_t flow_timeout, datapathid srcBr,
                            uint16_t srcOp, uint32_t srcOutLabel,
                            datapathid dstBr, uint16_t dstInPort);

// Setup an LSP - we do not support LSP paths with size < 2 - ie LSP must
// traverse at least 3 nodes including headend and tailend switch. We always do
// Penultimate Hop Popping due to lack of multiple tables in switch. Associate
// tunnelId tide with TE tunnel
bool setupLsp(Cspf::RoutePtr& route, tunn_elem& te, uint16_t payload_ethtype,
              tunnelId tid);

// LSP flow entries are removed in switches between the head and tail ends.
// flow entries in the head and tail ends will idle-timeout when the
// lsp is no longer used - we assume this is true when this function is called
bool teardownLsp(tunnelId tid, uint16_t payload_ethtype);


// To re-route flow over tunnel, sends a flow_mod message
// to head-end and possibly tail-end routers
void rerouteFlows(tunnelId newtid, datapathid srcBr, datapathid dstBr,
                  tunnelId oldtid);

// Route flow only over IP links (no tunnel-links)
void routeFlow(Flow& flow, datapathid br);

// Asks for flow stats from switch for the corresponding tunnel
// Does not request from tunnel head-end as we do not know what is matching
// and going into the tunnel. Thus it asks the switch that is the next-hop
```

```
// from the tunnel head-end.
void sendTunnelStatsReq(tunnelId tid);

// Given an explicit route in 'route', verifies that a route can be found
// along the explicit path which meets the bandwidth and priority constraints.
// Note that the caller may only specify the datapathids in 'route' and not the
// ports. In that case, the function populates the outport and inport for each
// link in the 'route'. If this route were to be installed, the caller MUST
// then submit a call to get_route. If this route could potentially eject
// some lower priority tunnels, then the tunnel_ids of those tunnels are reported
// in 'eject'. This function does NOT assume that the 'route' is eventually
// installed, nor does it assume the 'eject's are actually ejected.
// Finally, like get_route, this function is applicable only to routes for
// new tunnels, NOT existing ones. After checking, if the caller wishes to
// enable this new tunnel route, they can use get_route
bool check_explicit_route(RoutePtr& route, uint32_t resBw, uint8_t priority,
                          std::vector<uint16_t>& eject);

// Given a tunnel-id for an existing tunnel together with it's existing route
// (including port numbers), current reserved bandwdith and priority, this
// function checks if a new_reserved bandwidth is possible along the same route.
// If it is possible, the function returns true.
// If it is possible, but requires ejecting other lower priority tunnels, the
// function returns true and populates the would-be-ejected tunnel_ids
// in 'eject'.
// If it is possible, but requires using a new route, the function returns true,
// clears the 'route', populates the new 'route', and sets 'newroute' to true.
// If additionally the new 'route', bumps off lower priority tunnels, their
// tunnel_ids are returned in 'eject'
// This is a check - the function does not assume that the bandwidth reservation
// changes or new routes are installed or any tunnels are actually ejected.
bool check_existing_route(RoutePtr& route, uint32_t currResBw, uint8_t priority,
                          tunnelId tid, uint32_t newResBw, bool& newroute,
                          std::vector<uint16_t>& eject);

// In response to check_existing_route, the caller may decide to alter the
// existing route. The caller informs Cspf of the changes using
// set_existing_route. 'route is the existing route (when check_existing_route
// was called). if the route has since changed, then newroute is true and the new
// route is in 'new_route'. If the caller
// ejected routes to establish either the 'newResBw', or the 'new_route', it
// informs Cspf of the ejected tunnels in 'ejected'
void set_existing_route(uint16_t tid, uint8_t priority, uint32_t currResBw,
                        uint32_t newResBw, RoutePtr& route, bool newroute,
                        RoutePtr& new_route, std::vector<uint16_t>& ejected);

// rerouteEjectedTunnels is called whenever a call to Cspf::get_route returns
// tunnel ids of LSPs ejected due to CSPF routing with priorities. Tunnels may
// also be ejected as a result of auto-bandwidth. This function performs the
// following sequence of steps:
// 1) It asks Cspf::get_route for a new route for the ejected tunnel
// 2) It creates a new LSP over the new route and gives it a new tunnel-id and
// updates the Rib and tunnel_db with characteristics of the old ejected tunnel
// 3) moves flows from the ejceted-LSP to the newly created one
// 4) removes (un-installs) the ejected LSP from the data-plane
void rerouteEjectedTunnels(tunnId etid);
```