G E N I

Global Environment for Network Innovations

Milestone S2.j
Deliver Release of Measurement Handler

Document ID: GENI-MS2J-LEARN-May01

May 01, 2010

Prepared by:

D. Majumder[1], I. Baldine[3], D. Gurkan[1], M. S. Wang[2], C. P. Lai[2], K. Bergman[2]

1: University of Houston: College of Technology

2: Columbia University: Dept. of Electrical Engineering

3. RENCI, UNC-Chapel Hill

Under Project Nr. 1733

“Programmable Measurements over Texas-based Research Network: LEARN”

TABLE OF CONTENTS

1 Document Scope

This section describes this document's purpose, its context within the overall GENI project, the set of related documents, and this document's revision history.

1.1 Executive Summary

This technical note presents the results obtained in work package "Milestone S2.j: Deliver release of Measurement Handler" of Project Nr. 1733, "Programmable Measurements over Texas-based Research Network: LEARN" [LEARN_1].

This document is a release guide of the Measurement Handler Software (MHS). MHS is a set of Perl scripts that enables one to obtain real-time optical measurements (bit-error rate (BER) and optical power) from a variety of performance monitoring devices, including the Infinera Digital Transport Node (DTN) and the Polatis fiber switch. Further, IMF is a joint project among research teams from the University of Houston, University of North Carolina at Chapel Hill/RENCI, Columbia University, and North Carolina State University. At the University of Houston (GENI-LEARN), our task is to implement a MHS such that we can obtain BER and optical power measurements using the 4 DTNs located in the ORCA-BEN network. These measurements in turn would be used by the IMF project to further enable cross-layer experiments in optical networks. Measurement handlers for various embedded measurements in the GENI substrates and furthermore for some suggested external measurement instruments from our previous project (Data Plane Measurements [D_MEAS_1]) will be based on this first design and implementation of the MHS.

In Section 2, we summarize the work of our previous spiral 2 milestones. In Section 3, we provide the instructions for MHS. Section 4 gives a summary and conclusion.

1.2 Related Documents

The following documents are related to this document, and provide background information, requirements, etc., that are important for this document.

GENI Documents

Document ID	Document Title and Issue Date
GENI-S2H-LEARN-Nov09	Draft Measurement Data File Format, November 16, 2009
ERM_S2a_Dec09	Unified Measurement Framework: NetFPGA Cube Prototype, December 03, 2009
ERM_S2b_Mar10	Demonstrating Embedded Real-Time Physical Measurement from ORCA-BEN Substrate, March 04, 2010
GENI_MS2H_part2_LEARN_Jan08_msw	Specifications for the Measurement Handler Software, January 08, 2010
IMF Architecture	Milestone S2.d, IMF Project
GENI_MS2I_LEARN_Mar16	Implement and Integrate Measurement Handler, Mar 16, 2010

1.3 Document Revision History

The following table provides the revision history for this document, summarizing the date at which it was revised, who revised it, and a brief summary of the changes. This list is maintained in chronological order so the earliest version comes first in the list.

Revision	Date	Revised By	Summary of Changes
1.0	04/23/10	D. Majumder	Initial draft
1.1	04/23/10	D. Gurkan	Minor revision

2 Previous Work

The milestone on the drafting of the data file format for the transfer of measurement data between the Infinera Digital Transport Node (DTN) [Infinera_1, Infinera_2] and the Integrated Measurement Framework [IMF_1] was submitted during GEC6, November 16, 2009 [LEARN_2]. We explained how the measurement data file format is based on the TL1 command format of the Infinera DTN [Infinera_3, Infinera_4]. Also in [LEARN_2], we examined two specific TL1 commands that are of particular interest for the GENI measurement framework: an optical power measurement and the bit-error rate (BER) measurement. On January 8, 2010, a report on the specifications for the Measurement Handler Software was submitted. The implementation report of the software was presented on March 16, 2010 [LEARN_3].

3 Measurement Handler Software Release Guide

The Measurement Handler Software (MHS) is a server program, which communicates with Integrated Measurement Framework (IMF) using the XML-RPC protocol. MHS supports invocation of three procedures/functions:

a. connectToDevice – logs into the device (Infinera/Polatis),

b. retrieveMeasurement – fetch measurement from device,

c. disconnectFromDevice – logs out from the device.

It supports multiple logins as well as the simultaneous retrieval of measurements. The following measurement types are currently supported:

- Infinera: PREFEC-BER, POSTFEC-BER and PORTPOWER (received power),
- Polatis: PORTPOWER.

3.1 Installation Procedure

Keep the following files in the same folder:

- tl1.pm

- pol_tl1.pm
- infinera_tl1.pm
- XMLRPC_API_Server_MH_InfineraPolatis.pl

Install Perl (minimum version 5.6.0) and the following Perl modules from CPAN [http://www.cpan.org/]:

- threads;
- threads::shared;
- Thread::Queue::Any;
- Net::Telnet;
- Date::Manip;
- CGI::Cookie;
- SOAP::Lite;
- Thread::Semaphore;

Run the script as:

```
perl XMLRPC_API_Server_MH_InfineraPolatis.pl
```

The script polls the port 8001 for measurement requests from clients. It prints the login, logout, and error details with the corresponding time-stamps.

Screenshot of sample output:

```
Contact to XMLRPC server at http://geni-imf-dev:8001
2010-03-09 19:11:48: ID1001-Logged into dtn-1.duke.ben Infinera.
2010-03-09 19:11:48: ID1002-Logged into os.duke.ben Polatis.
2010-03-09 19:29:53: ID1001-Logged out from dtn-1.duke.ben Infinera.
2010-03-09 19:29:53: ID1002-Logged out from os.duke.ben Polatis.
```

3.2 Invoking the server methods from client software (such as the IMF)

3.2.1 Login Procedure

For a connectToDevice call, the request should have the format:

```
<?xml version="1.0" encoding="UTF-8"?> <methodCall>
<methodName>connectToDevice</methodName>
<params>parameters</params>
</methodCall>
```

where the parameters/arguments are:

- `deviceIP=<IP address or FQDN of device>`
- `port=9090/3082`
- `username=<username>`
- `password=<pwd>`
- `device=Infinera/Polatis` (case-insensitive)

Using these parameters, a new tl1 connection is created with the device and a unique connection identifier is returned to the client in the XML-RPC structure:

```
<?xml version="1.0" encoding="UTF-
8"?><methodResponse><params><param><value><struct>
<member><name>connectionObject</name><value><int>UNIQUE
ID</int></value></member>
</struct></value></param></params></methodResponse>
```

In case of login failure, the appropriate error message is returned. The procedure can accept concurrent requests. Every connection is automatically logged off if it is idle for more than 5 minutes.

3.2.2 Measurement Retrieval Procedure

For a retrieveMeasurement call, the request should have the format:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall><methodName>retrieveMeasurement</methodName>
<params>PARAMETERS</params>
</methodCall>
```

where the following parameters are expected:

- `connectionObject=ID(Id returned by the connectToDevice method)`
- `chassis=<val> DLMslot=<val> opticalChannel=<val> (for Infinera)`
- `port=<val> (for Polatis)`
- `measureType=prefec-ber/postfec-ber/portpower` (for Infinera) `portPower` (for Polatis)

The measureType is case-insensitive.

In the case of successful retrieval, the result is returned in the format:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse><params><param><value><struct><member>
```

```
<name>MEASURETYPE</name><value><double>MEASUREMENT_VALUE</double></va
lue>
</member></struct></value></param></params></methodResponse>
```

The function can fetch concurrent requests from the same as well as different connection ids. In the case of failure, an appropriate error message is returned with the <name> field as ERROR instead of MEASURETYPE.

3.2.3 Logout Procedure

For a disconnectToDevice call, the request should have the format:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodCall><methodName>disconnectFromDevice</methodName>
<params>PARAMETERS</params></methodCall>
```

where the parameters are:

- `connectionObject=ID (Id returned by the connectToDevice method)`
- `deviceIP=<IP address or FQDN of device>`
- `port=9090/3082`
- `username=<username>`
- `password=<pwd>`
- `device=Infinera/Polatis`

The username and password is required to prevent erroneous disconnections. For successful disconnection, the response will be sent in the below format:

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse><params><param><value><struct><member>
<name>Disconnected</name><value><int>ID</int></value>
</member></struct></value></param></params></methodResponse>
```

Concurrent disconnection requests can be processed.

3.2.4 Remarks

For any of the above requests, if the device (Infinera/Polatis) does not respond within 200 seconds (3min 20 sec), the request is canceled and an error message is returned.

4 Summary and Conclusions

In this report, we drafted the release guide for the Measurement Handler Software. We previously defined the measurement data file format as the TL1 response format of the Infinera DTN and the specifications for the measurement handler software. We also documented the specifications for integrating MHS with IMF.

5 Bibliography

[1] [LEARN_2] D. Gurkan, Y. Xin, M. S. Wang, C. P. Lai, and K. Bergman, “Specifications for the Measurement Handler Software” [Online]. Available: http://groups.geni.net/geni/attachment/ticket/270/GENI_MS2H_part2_LEARN_Jan08.pdf
[2] [D_MEAS_1] Data Plane Measurements: Milestone report, Identify external measurement equipment [Online]. Available: http://groups.geni.net/geni/attachment/wiki/Data%20Plane%20Measurements/GENI_MS3_DMEAS.docx
[3] [DMEAS_2] Data Plane Measurements: DMEAS [Online]. Available: http://groups.geni.net/geni/wiki/Data%20Plane%20Measurements
[4] [LEARN_1] Programmable Measurements over Texas-based Research Network: LEARN [Online]. Available: http://groups.geni.net/geni/wiki/LEARN
[5] [LEARN_3] D. Majumder, M. S. Wang, I. Baldine, D. Gurkan, C. P. Lai, K. Bergman, “Implement and Integrate Measurement Handler” [Online]. Available: http://groups.geni.net/geni/attachment/wiki/LEARN/GENI_MS2I_LEARN_Mar16.pdf
[6] [ERM_1] C. P. Lai, M. S. Wang, K. Bergman, “Unified Measurement Framework: NetFPGA Cube Prototype,” Dec. 2009 [Online]. Available: http://groups.geni.net/geni/attachment/ticket/279/ERM_S2a_Dec09.pdf
[7] ERM_2] M. S. Wang, C. P. Lai, and K. Bergman, “Demonstrating Embedded Real-Time Physical Measurement from ORCA-BEN Substrate”, Mar. 2010 [Online]. Available: http://groups.geni.net/geni/attachment/wiki/Embedded%20RealTime%20Measurements/ERM_S2b_Mar10.pdf?format=raw
[8] [imf_1] I.Baldine, K.Bergman, R.Dutta, D.Gurkan, C.Lai, G.Rouskas, A. Wang, M.S. Wang, “IMF Architecture (Project Nr. 1718, Milestone S2.d),” (2010, Feb) [Online]. Available: https://geni-imf.renci.org/trac/wiki
[9] [Infinera_1] “Infinera DTN Hardware Description Guide,” Release 5.0, Version 005, Document ID 1900-215.
[10] [Infinera_2] Infinera DTN [Online]. Available: http://www.infinera.com/products/DTN.html
[11] [LEARN_2] M. Wang, D. Gurkan, C. P. Lai, K. Bergman, “Draft Measurement Data File Format,” November 2009 [online]. Available: http://groups.geni.net/geni/attachment/ticket/270/GENI_S2H_LEARN_Nov09.pdf
[12] [SARA_1] R. van der Pol, A. Toonk, “Data Gathering in Optical Networks with the TL1 Toolkit,” Passive and Active Measurements Conference (PAM), 2009.
[13] [xmpp_1] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core
[14] [xmpp_2] RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence
[15] [xmpp_3] XEP-0060: Publish Subscribe (http://xmpp.org/extensions/xep-0060.html)
[16] [xmpp_4] OpenFire XMPP server (http://www.igniterealtime.org/projects/openfire/)

6 Appendix

6.1 Perl Modules

tl1.pm (Modified code of tl1.pm from SARA High Performance Networking. Copyright (c) 2005 - 2007)

```
package tl1;

use strict;
use warnings;
# Module import
use threads;
use threads::shared;
use Thread::Queue::Any;
use Net::Telnet;

my %cardtype;

# Module export
our @EXPORT  = qw(new);

our $VERSION = "1.3.1";

#my $keep_reading : shared = 1;
my %keep_reading : shared; #Changed to hash to support multiple logins

# Constructor method
sub new {
    my $invocant = shift;
    my $class    = ref($invocant) || $invocant;
    my $connectionId= shift;
    $keep_reading{$connectionId} = 1;#Changed to hash to support multiple logins
    my $self     = {
            username => 'xxxx',
            password => 'xxxx',
            type => 'OME6500',
            ctag => '1',
            timeout => 10,
            retries => 3,
            verbose  => 0,
            connectionId => $connectionId,
            @_,
    };

    bless( $self, $class );
    # 3 threads started
```

```
        #resp_queue has responses
        $self->{'resp_queue'} = Thread::Queue->new();
        #auto_queue has autonomous messages
        $self->{'auto_queue'} = Thread::Queue->new();
        #
        $self->{'acmd_queue'} = Thread::Queue->new();

        $self->verbose( 2,"Created tl1 object for: $self->{'hostname'}.\n" );
        return $self;
}

sub verbose {
        my ( $self, $level, $info ) = @_;
        print( STDERR "debug$level: $info" )
          if ( $level <= $self->{'verbose'} );
}

sub receiver_thread {
        my ($self) = @_;      # Object this method is part of
        my $state  = 'noop';    # State of FSM
        my $n      = 0;                 # Counter variable
        my @response;                   # Local copy of return array
        my $socket = $self->{'socket'};    # Socket

        # Debug message
        $self->verbose( 3, "Starting receiver thread.\n" );

        my $buf;
        while ($keep_reading{$self->{'connectionId'}}) {
                # The OM5200 does not end its response with a newline.
                # Therefore, the last line is a single ';' character.
                # Look ahead in the buffer to detect this situation.
                my $bufref = $socket->buffer();
                if (!defined($$bufref)) {
                        $self->verbose(1, "bufref not defined\n");
                }
                if ((length($$bufref) == 1) && ($$bufref eq ";") &&
                                (($state eq 'response') || $state eq 'auto')) {
                        $self->verbose( 9, "; read, state == response\n" );
                        $buf = $socket->get();
                } else {
                        $self->verbose( 9, "state = $state\n" );
                        $buf = $socket->getline();
                }
                if (!defined($buf)) {
                        $n++;
                        if (!$socket->eof()) {
                                my $msg = $socket->errmsg;
                                if ($msg ne "") {
                                        $self->verbose(1, "receiver_thread $self->{'hostname'} (state=$state,
retries=$n): $msg\n");
                                }
                        }
                        # not sure if @resp needs to be shared
                        #my @resp : shared = undef;
                        if ($n >= $self->{'retries'}) {
                                $self->{'resp_queue'}->enqueue(undef);
                                $n = 0;
                        }
                        next;
                }
                # Debug message
                $self->verbose( 3, "$buf\n" );
                $n = 0;
        SWITCH: {
                        # a line with a ; only means the end of a response
                        $buf =~ /^;/ && ( $state =~ /auto|response/ ) && do {
                                if ( $state eq 'auto'
                                        && defined( $self->{'auto_code'} ) )
                                {

                                        # Return autonomous data
                                        my @resp : shared = @response;
                                        $self->{'auto_queue'}->enqueue( \@resp );
                                }
                                elsif ( $state eq 'response' ) {
                                        # this is what we do at the end of a response
                                        # Return response data in @response
                                        # it returns a reference to @response called  @resp

                                        my @resp : shared = @response;
                                        $self->{'resp_queue'}->enqueue( \@resp );
                                }
                                @response = ();
                                $socket->buffer_empty();

                                $state = 'noop';
                                last SWITCH;
                        };

                        $buf =~ /^[;\r\n\<\>]/ && do {
                                $state = 'noop';
```

```
                    last SWITCH;
                };

                # start of an autonomomous message (alarm)
                $buf =~ /^(\*C|\*\*|\* |A ) (\d{1,10}) (.*)/ && do {
                    $state = 'auto';
                    last SWITCH;
                };

                # start of a response
                $buf =~ /^M  (.*) (COMPLD|PRTL|DENY)/ && do {
                    $state = 'response';
                    last SWITCH;
                };

                # line with host date and time
                $buf =~ /(   (.*) (\d{2}-\d{2}-\d{2}) (\d{2}:\d{2}:\d{2}))/
                 && do {
                    $state = 'log';
                    last SWITCH;
                 };
            }
            if ($state ne 'noop') {
                chomp($buf);
                # lines are pushed in @response until a line with
                # a ; only ($state equals 'noop') is read
                push( @response, $buf );
            }
        }
        $self->{'resp_queue'}->enqueue(undef);
        $self->verbose( 3, "Receiver thread ended.\n" );
}

sub auto_thread {
        my ($self) = @_;

        $self->verbose( 3, "Autonomous messages thread started.\n" );
        while ( my $msg_ref = $self->{'auto_queue'}->dequeue() ) {
            $self->{'auto_code'}->($self, $msg_ref,
                                    $self->{'auto_arg_ref'} );
        }
        $self->verbose( 3, "Autonomous messages thread ended.\n" );
}

sub set_auto_code {
        my ( $self, $func_ref, $arg_ref ) = @_;
        $self->{'auto_code'}    = $func_ref;
        $self->{'auto_arg_ref'} = $arg_ref;
}

sub async_cmd_thread {
        my ($self) = @_;

        $self->verbose( 3, "Starting async command thread.\n" );
        while ( my $cmd = $self->{'acmd_queue'}->dequeue() ) {
                my $resp_ref = $self->cmd($cmd);
                $self->{'async_code'}->( $resp_ref, $self->{'async_arg_ref'} )
                  if defined( $self->{'async_code'} );
        }
        $self->verbose( 3, "Async command thread ended.\n" );
}

sub set_async_code {
        my ( $self, $func_ref, $arg_ref ) = @_;
        $self->{'async_code'}    = $func_ref;
        $self->{'async_arg_ref'} = $arg_ref;
}


sub open {
        my ($self) = @_;
        $self->verbose( 3,
                "Starting threads for tl1 device: $self->{'hostname'}.\n" );

        # open socket connection
        my $con = 1;
        $self->{'socket'} = new Net::Telnet(
                Timeout    => $self->{'timeout'},
                Errmode    => 'return'
        );
        $self->{'socket'}->port($self->{'peerport'});
        $self->{'socket'}->open($self->{'hostname'});

        if ( $con != 1 ) {
                $self->verbose(1, "Can't connect to $self->{'hostname'}\n");
                return 0;
        }

        $self->set_auto_code(\&deviceError,\'Logging Off!' );

        # Handle alarms
        if ( defined( $self->{'auto_code'} ) ) {
```

```
                        $self->{'auto_thread'} = async {
                                $self->auto_thread();
                         }
                }

                # Start receiver
                $self->{'receiver_thread'} = async {
                        $self->receiver_thread();
                };

                # Async commands
                $self->{'async_cmd_thread'} = async {
                        $self->async_cmd_thread();
                };

                # login
                my $result = $self->login();
                if ($result ne "COMPLD") {
                        $self->verbose(1, "login failed for $self->{'hostname'} $self->{'username'}\n");
                        $self->{'socket'}->close();
                        if ( defined( $self->{'auto_code'} ) ) {
                                $self->{'auto_queue'}->enqueue(undef);
                                $self->{'auto_thread'}->join();
                        }
                        $self->{'acmd_queue'}->enqueue(undef);
                        $self->{'async_cmd_thread'}->join();
                        $keep_reading{$self->{'connectionId'}}= 0;
                        $self->{'receiver_thread'}->join();
                        return 0;
                } else {
                        $self->verbose( 2, "login succeeded $self->{'username'}.\n" );
                }
                return 1;

        }



        # Closes connection
        sub close {
                my ($self) = @_;
                $self->verbose( 2, "Closing connection to $self->{'hostname'}.\n" );
                my $socket = $self->{'socket'};
                $socket->print("canc-user::$self->{'username'}:$self->{'ctag'};\n");
                $keep_reading{$self->{'connectionId'}}= 0;
                my $out = $self->{'resp_queue'}->dequeue();
                my $check = check_resp($out);
                $self->{'socket'}->close();
                if ( defined( $self->{'auto_code'} ) ) {
                        $self->{'auto_queue'}->enqueue(undef);
                        $self->{'auto_thread'}->join();
                }
                $self->{'acmd_queue'}->enqueue(undef);
                $self->{'async_cmd_thread'}->join();
                $self->{'receiver_thread'}->join();
                return $check;
        }

        sub login() {
                my ($self) = @_;
                $self->verbose( 3, "starting login to $self->{'hostname'}.\n" );
                my $out = $self->cmd("ACT-USER::$self->{'username'}:$self->{'ctag'}::$self->{'password'};\n");
                my $check = check_resp ($out);
                return $check;
        }


        # Send TL1 command and wait for response
        sub cmd {
                my ( $self, $cmd ) = @_;
                my $socket = $self->{'socket'};
                $socket->print($cmd);
                my $resp_ref = $self->{'resp_queue'}->dequeue();
                return $resp_ref;
        }


        #
        # return card type (CTYPE) string present in $shelf/$slot
        #
        sub get_cardtype {
                my ($self, $shelf, $slot) = @_;

                my $socket = $self->{'socket'};
                $socket->print("RTRV-INVENTORY::SLOT-$shelf-$slot:$self->{'ctag'};\n");
                my $resp = $self->{'resp_queue'}->dequeue();
                my $n = 0;
                while (defined($resp->[$n])) {
                        if ($resp->[$n] =~ /"AP-\d+-\d+::CTYPE/) {
                                # skip Access Panel line
                        } elsif ($resp->[$n] =~ /"\S+-\d+-\d+::CTYPE=\\"(.*?)\\",/) {
                                return $1;
                        }
                        $n++;
                }
                return undef();
        }


        sub retr_cardtype {
                my ( $self  ) = @_;
                my $socket = $self->{'socket'};
                $self->verbose( 1, "detecting card type\n" );
                $socket->print("RTRV-EQPT::ALL:$self->{'ctag'};\n");
                my $resp = $self->{'resp_queue'}->dequeue();
                #return $resp;
                my $n = 0;
                while (defined($resp->[$n])) {
                # match "PGEFC-1-1-3::PROVPEC=NTTP01AF,CTYPE=\"GE SX/FC100 Mx 850nm LC
SFP\",PEC=NTTP01AF,REL=01,CLEI=SOC3NNFCAA,SER=AGLTMY00000VZG  :IS,IDLE"
                # match "10GW-1-6::PROVPEC=NTK526KE,CTYPE=\"1xOC-192/STM-64 STS-1/HO DWDM
1530.33nm\",PEC=NTK526KE,REL= 05 bablabla
                # match L2SS-1-12::PROVPEC=NTK531BA,CTYPE=\"4xGE L2SS 2.5G VT1.5/LO SFP\",PEC=NTK531BA,REL= 01
                        if ($resp->[$n] =~ /"(\w+)-1-(\d+)::.*"/) {
                                $cardtype {$2} = $1;
                        }
                        $n++;

                }
                return %cardtype;
        }


        sub retr_swversion {
                my ( $self  ) = @_;
                my $swversion;
                #$self->verbose( 1, "retr software vesion\n" );
                my $socket = $self->{'socket'};
                if ($self->{'type'} eq "OME6500") {
                        $socket->print("RTRV-SW-VER:::$self->{'ctag'};\n");
                        my $resp = $self->{'resp_queue'}->dequeue();
                        my $n = 0;
                        while (defined($resp->[$n])) {
                                # SHELF-1:REL0200Z.HQ
                                if ($resp->[$n] =~ /"SHELF-1:(\w+).*"/) {
                                    $swversion = $1;
                                }
                                $n++;
                        }
                } elsif ($self->{'type'} eq "HDXc") {
                        $socket->print("RTRV-UPGRD-STATUS:::$self->{'ctag'};\n");
                        my $resp = $self->{'resp_queue'}->dequeue();
                        my $n = 0;
                        while (defined($resp->[$n])) {
                                # "RELEASE=HDX_Rel_03.30_fo09,...
                                if ($resp->[$n] =~ /"RELEASE=([^,]+),/) {
                                    $swversion = $1;
                                }
                                $n++;
                        }
                }
                return $swversion;
        }


        sub retr_inoctets {
                my ( $self  ) = @_;
                my $inoctets = 0;
                $self->verbose( 1, "retr in octets\n" );
                my $socket = $self->{'socket'};
                my $slot= $self->{'slot'};
                my $port = $self->{'port'};
                my $swver = $self->retr_swversion;
                if (!defined($swver)) {
                        return undef;
                }
        #       if ($swver eq 'REL0200Z') {
        #               $self->verbose( 1, "sw version is REL0200Z  command RTRV-OM-ETH::ETH-1-$slot-$port:45::;\n" );
        #               $socket->print("RTRV-OM-ETH::ETH-1-$slot-$port:$self->{'ctag'}::;\n");
        #       } else  {
        #               $socket->print("RTRV-OM-if::ETH-1-$slot-$port:$self->{'ctag'}::;\n");
        #       }
                $socket->print("RTRV-OM-ETH::ETH-1-$slot-$port:$self->{'ctag'}::;\n");
                my $resp = $self->{'resp_queue'}->dequeue();
                my $n = 0;
                while (defined($resp->[$n])) {
                #match ETH-1-4-
1::INFRAMES=343136315745,INFRAMESERR=0,INOCTETS=21960686905538,INDFR=1023574,INFRAMESDISCDS=0,INCFR=0,F
RTOOSHORTS=0,FCSERR=0,FRTOOLONGS=0,OUTFRAMES=948548260,OUTOCTETS=68296800619,OUTPAUSEFR=0,OUTDFR=0
,INTERNALMACRXERR=0,INTERNALMACTXERR=0,DEFERTRANS=77813222
                        if ($resp->[$n] =~ /".*,INOCTETS=(\d+).*"/) {
                                $inoctets = $1;
                        }
                        $n++;
                }
                return $inoctets;
        }


        sub retr_outoctets {
                my ( $self  ) = @_;
                my $outoctets = 0;
                $self->verbose( 1, "retr out octets\n" );
                my $socket = $self->{'socket'};
                my $slot= $self->{'slot'};
                my $port = $self->{'port'};
                my $swver = $self->retr_swversion;
                if (!defined($swver)) {
                        return undef;
                }
        #       if ($swver eq 'REL0200Z') {
        #               $self->verbose( 1, "sw version is REL0200Z  command RTRV-OM-ETH::ETH-1-$slot-$port:45::;\n" );
        #               $socket->print("RTRV-OM-ETH::ETH-1-$slot-$port:$self->{'ctag'}::;\n");
        #       } else  {
        #               $socket->print("RTRV-OM-if::ETH-1-$slot-$port:$self->{'ctag'}::;\n");
        #       }
                $socket->print("RTRV-OM-ETH::ETH-1-$slot-$port:$self->{'ctag'}::;\n");
                my $resp = $self->{'resp_queue'}->dequeue();
                my $n = 0;
                while (defined($resp->[$n])) {
                #matchETH-1-4-
1::INFRAMES=343136315745,INFRAMESERR=0,INOCTETS=21960686905538,INDFR=1023574,INFRAMESDISCDS=0,INCFR=0,F
RTOOSHORTS=0,FC
SERR=0,FRTOOLONGS=0,OUTFRAMES=948548260,OUTOCTETS=68296800619,OUTPAUSEFR=0,OUTDFR=0,INTERNALMACRXE
RR=0,INTERNALMACTXERR=0,DEFERTRANS=77813222
                        if ($resp->[$n] =~ /".*,OUTOCTETS=(\d+).*"/) {
                                $outoctets = $1;
                        }
                        $n++;
                }
                return $outoctets;
        }


        # alarms
        sub retrieve_alarms {
                my ($self) = @_;
                my $key;
                my @result;
                my $ainfo;
                my $val;
        my $socket = $self->{'socket'};
        $self->verbose( 1, "retrieving alarms for $self->{'type'}\n" );
        if ($self->{'type'} eq "OME6500") {
                $socket->print("RTRV-ALM-ALL:::$self->{'ctag'}:::;\n");
                my $resp = $self->{'resp_queue'}->dequeue();
                my $n = 0;
                while (defined($resp->[$n])) {
                    # match "LAN-1-15,COM:MN,NET,NSA,11-01,15-44-31,NEND,NA:\"LAN-15 Port
Failure\":0100000002-5003-0536,:YEAR=2005,MODE=NONE"
                    if ($resp->[$n] =~ /"([^,]+),\w+:(\w+),\w+,(\w+),(\d+-\d+),(\d{1,2}-\d{1,2}-
\d{1,2}),.*:\\"(.*)\\".*YEAR=(\d\d\d\d),.*"/) {
                        #print "$1 $2 $3 $4 $5 $6 $7\n";
                        my $interface = $1;
                        my $severity = $2;
                        my $impact = $3;
                        my $day = $4;
                        my $time = $5;
                        my $alarm = $6;
                        my $year = $7;
                        my $date = "$7-$4";
                        my $slot = undef;
                        my $subslot = undef;
                        my $port = undef;
                        my $fromfirststs = undef;
                        # interface can have diferent forms:
                        # STS3C-1-5-1-88
                        # WAN-1-2-4
                        # ETH-1-1-2
                        # 10G-1-10
                        # FILLER-1-9
                        if ($interface =~ /\w+-\d+-(\d+)-?(\d+)?-?(\d+)?/) {
                            $slot = $1;
                            if (defined $2){
                                #print "port is $2\n";
                                $port = $2;
                            }
                            if (defined $3){
                                #print "beginsts is $3\n";
                                $fromfirststs = $3;
                            }
                        }

                        #push @result, [$interface, $slot, $subslot, $port, $fromfirststs, $severity, $impact,
$date, $time, $alarm ];
                        $ainfo = {
                            interface => $interface,
                            slot => $slot,
                            subslot => $subslot,
                            port => $port,
                            fromfirststs => $fromfirststs,
                            severity => $severity,
                            impact => $impact,
                            date => $date,
                            time => $time,
                            alarm => $alarm
                        };
                        push @result, $ainfo;
                    }
                    $n++;
                }
                return @result;

        } elsif ($self->{'type'} eq "HDXc") {
                $socket->print("RTRV-ALM-ALL:::$self->{'ctag'}:::;\n");
                my $resp = $self->{'resp_queue'}->dequeue();
                my $n = 0;
                #print_resp($resp);
                while (defined($resp->[$n])) {
                    # match "ESI-2-SATT-1,TMG:MN,SYNCLOS,NSA,09-13,04-18-28,NEND,RCV:\"Loss of
signal\",,\"TYP-SH-SATT-PRT\",:000006-6040-0066,:YEAR=2005"
                    if ($resp->[$n] =~ /"([^,]+),\w+:(\w+),\w+,(\w+),(\d+-\d+),(\d{1,2}-\d{1,2}-
\d{1,2}),.*:\\"(.*)\\",.*,.*,.*:YEAR=(\d\d\d\d).*"/) {
                        #print "$1 $2 $3 $4 $5 $6 $7\n";
                        my $interface = $1;
                        my $severity = $2;
                        my $impact = $3;
                        my $day = $4;
                        my $time = $5;
                        my $alarm = $6;
                        my $year = $7;
                        my $date = "$7-$4";
                        my $slot = undef;
                        my $subslot = undef;
                        my $port = undef;
                        my $fromfirststs = undef;

                        # Interface format HDXc is:
                        # Payload AID:
                        # <type> <shelf> <slot> <subslot> <port> <signal> <offset>
                        # signal seems to be always 1, not sure what this means.
                        # offset is STS channel
```

```
                                        if ($interface =~
                                                  /^\w+-\d+-(\d+)-(\d+)-(\d+)-(\d+)-(\d+)/) {
                                                  $slot = $1;
                                                  $subslot = $2;
                                                  $port = $3;
                                                  $fromfirststs = $5;
                                        } elsif ($interface =~
                                                  /^\w+-\d+-(\d+)-(\d+)-(\d+)-(\d+)/) {
                                                  $slot = $1;
                                                  $subslot = $2;
                                                  $port = $3;
                                        } elsif ($interface =~
                                                            /^\w+-\d+-(\d+)-(\d+)/) {
                                                  $slot = $1;
                                                  $port = $2;
                                        } elsif ($interface =~ /^\w+-\d+-(\d+)/) {
                                                  $slot = $1;
                                        }
                                        #push @result, [$interface, $slot, $subslot, $port, $fromfirststs, $severity, $impact,
$date, $time, $alarm ];
                                        $ainfo = {
                                                  interface => $interface,
                                                  slot => $slot,
                                                  subslot => $subslot,
                                                  port => $port,
                                                  fromfirststs => $fromfirststs,
                                                  severity => $severity,
                                                  impact => $impact,
                                                  date => $date,
                                                  time => $time,
                                                  alarm => $alarm
                                        };
                                        push @result, $ainfo;
                              }
                              $n++;
                    }
                    return @result;
          } elsif ($self->{'type'} eq "ONS15454") {
                    $socket->print("RTRV-ALM-ALL:::$self->{'ctag'};\n");
                    my $resp = $self->{'resp_queue'}->dequeue();
                    my $n = 0;
                    #print_resp($resp);
                    while (defined($resp->[$n])) {
                              # match FAC-3-1,G1000:MN,CARLOSS,NSA,11-23,16-43-47,,:\"Carrier Loss On The
LAN\",G1000-4"
                              # FAC-12-1,OC19:MN,LO-RXPO,NSA,11-01,06-49-20,,:\"Equipment Low Rx power\",OC192"
```

```
if ($resp->[$n] =~ /"([^,]+),.*:(\w+),.*,(\w+),(\d+-\d+),(\d{1,2}-\d{1,2}-\d{1,2}),[^:]:\\"(.*)\\"/) {
        # print "$1 $2 $3 $4 $5 $6 $7\n";
        my $interface = $1;
        my $severity = $2;
        my $impact = $3;
        my $day = $4;
        my $time = $5;
        my $alarm = $6;
        my $slot = undef;
        my $subslot = undef;
        my $port = undef;
        my $fromfirststs = undef;

        # ONS does not report the year so we have to create this by ourselfs
        use Time::localtime;
        use Date::Manip;
        my $year = localtime->year() + 1900;
        my $date = "$year-$day";
        # now we need to test if the date is not a day in the future
        my $testdate = ParseDate("$date");
        my $todaydate = ParseDate("today");
        my $flag = Date_Cmp($testdate,$todaydate);
        if ($flag<0) {
                #print " date1 is earlier\n";
        } elsif ($flag==0) {
                #print " the two dates are identical\n";
        } else {
                #print " testdate is earlier, so adjust to last year\n";
                my $newdate = DateCalc("$testdate","-1 Year",\my $err);
                $date = UnixDate("$newdate","%Y-%m-%d");
                #print "$testdate\n";
        }

        # FAC-6-1  FAC-12-1
        if ($interface =~ /\w+-(\d+)-?(\d+)?-?(\d+)?/) {
                $slot = $1;
                if (defined $2){
                        #print "port is $2\n";
                        $port = $2;
                }
                if (defined $3){
                        #print "beginsts is $3\n";
                        $fromfirststs = $3;
                }
        }
```

```
                                        #push @result, [$interface, $slot, $subslot, $port, $fromfirststs, $severity, $impact,
$date, $time, $alarm ];
                                        $ainfo = {
                                                interface => $interface,
                                                slot => $slot,
                                                subslot => $subslot,
                                                port => $port,
                                                fromfirststs => $fromfirststs,
                                                severity => $severity,
                                                impact => $impact,
                                                date => $date,
                                                time => $time,
                                                alarm => $alarm
                                        };
                                        push @result, $ainfo;
                                }
                                $n++;
                        }
                        return @result;
                }
        }


        # Retrieve circuits from host and store in circuits data structure
        sub retrieve_circuits {
                my ($self) = @_;
                my %sts;
                my $key;
                my @result;
                my $val;
                my $socket = $self->{'socket'};
                $self->verbose( 1, "retrieving circuits for $self->{'type'}\n" );
                if ($self->{'type'} eq "OME6500") {
                        # First determine which cards are there
                        my %cards = $self->retr_cardtype;
                        #my %cards = retr_cardtype();
                        $socket->print("RTRV-CRS-COUNT:::$self->{'ctag'}::;\n");
                        my $resp = $self->{'resp_queue'}->dequeue();
                        my $n = 0;
                        while (defined($resp->[$n])) {
                                # match "SHELF-1:STS3C,8"
                                # first we need to determine which crossconnects are there (sts3, sts24, sts192 etc..)
                                if ($resp->[$n] =~ /SHELF-\d+:(STS\w+),\d+/) {
                                        $sts{$1} = 1;
                                }
                                $n++;
```

```
		}
		# now for each type of crossconnect retrieve the info
		foreach $key (keys %sts) {
			$socket->print("RTRV-CRS-".$key."::ALL:$self->{'ctag'}:::DISPLAY=PROV,CKTID=ALL;\n");
			my $resp = $self->{'resp_queue'}->dequeue();
			my $n = 0;
			while (defined($resp->[$n])) {
				# match "STS3C-1-9-1-1,STS3C-1-4-101-1:2WAY:CKTID=\"test\":"
				# match "STS3C-1-9-1-19,STS3C-1-2-1-19:2WAYPR:SWMATE=STS3C-1-6-1-
40,CKTID=\"Ah001A-Es001A_GE1(Artez) \":"
				if ($resp->[$n] =~ /(STS\w+)-(\d+)-(\d+)-(\d+)-(\d+),STS\w+-(\d+)-(\d+)-(\d+)-
(\d+):(2WAY:|2WAYPR:SWMATE=STS\w+)?-?(\d+)?-?(\d+)?-?(\d+)?-?(\d+)?.*CKTID=\\"([^:]+)\\":"/) {

					my $id = $15;
					my $bandwidth = $1;
					my $speed = $bandwidth;
					my $shelf = $2;
					my $beginslot = $3;
					my $beginsubslot = '';
					my $beginport = $4;
					my $beginfromsts;
					my $begintosts;
					my $endslot = $7;
					my $endsubslot = '';
					my $endport = $8;
					my $endfromsts;
					my $endtosts;
					my $swmateslot = $12;
					my $swmatesubslot = '';
					my $swmateport = $13;
					my $swmatefromsts;
					my $swmatetosts;

					$beginfromsts = $5;
					$self->verbose( 1, "from Card is $cards{$beginslot} and beginfrom sts is
$beginfromsts\n" );

					$endfromsts = $9;
					$self->verbose( 1, "to Card is $cards{$endslot} and tofrom sts is
$endfromsts\n" );

					if (defined $swmateslot) {
						$swmatefromsts = $14;
						$speed =~ s/\D//g;
						$swmatetosts = $swmatefromsts + $speed - 1;
					}
```

```
                                                $speed =~ s/\D//g;
                                                $begintosts = $beginfromsts + $speed - 1;
                                                $endtosts = $endfromsts + $speed - 1;

                                                # catch undefined values
                                                if (not defined $swmateslot) {
                                                        $swmateslot = '';
                                                }
                                                if (not defined $swmateport) {
                                                        $swmateport = '';
                                                }
                                                if (not defined $swmatefromsts) {
                                                        $swmatefromsts = '';
                                                }
                                                if (not defined $swmatetosts) {
                                                        $swmatetosts = '';
                                                }
                                                push @result, [$id, $speed, $beginslot, $beginsubslot, $beginport,
$beginfromsts, $begintosts, $endslot, $endsubslot, $endport, $endfromsts, $endtosts, $swmateslot, $swmatesubslot,
$swmateport, $swmatefromsts, $swmatetosts];
                                        }
                                        $n++;
                                }
                        }
                        #print "$val\n" while defined($val = pop(@result));
                } elsif ($self->{'type'} eq "HDXc") {
                        # Voor de HDXc
                        $socket->print("RTRV-CRS-ALL:::$self->{'ctag'};\n");
                        my $resp = $self->{'resp_queue'}->dequeue();
                        my $n = 0;
                        #print_resp($resp);
                        while (defined($resp->[$n])) {
                                # match  "OC192-1-503-0-1-1-49,OC192-1-505-0-3-1-49:2WAY,STS-
48C:PRIME=OSS,DISOWN=IDLE,CONNID=881,LABEL=\"SC05-Prague-Chicago-1\",AST=LOCKED:ACT"
                                if ($resp->[$n] =~ /[^-]+-(\d+)-(\d\d\d)-(\d)-(\d)-\d{1,3}-(\d{1,3}),[^-]+-\d+-(\d\d\d)-(\d)-(\d)-
\d{1,3}-(\d{1,3}):[^,]+,([^:]+):.*LABEL=\\\"(.*)\\\",/) {
                                        my $id = $11;
                                        my $bandwidth = $10;
                                        my $shelf = $1;
                                        my $beginslot = $2;
                                        my $beginsubslot = $3;
                                        my $beginport = $4;
                                        my $beginfromsts = $5;
                                        my $endslot = $6;
                                        my $endsubslot = $7;
```

```
				my $endport = $8;
				my $endfromsts = $9;
				my $speed = $bandwidth;
				# \D remove all non digits
				$speed =~ s/\D//g;
				my $begintosts = $beginfromsts + $speed - 1;
				my $endtosts = $endfromsts + $speed - 1;

				push @result, [$id, $speed, $beginslot, $beginsubslot, $beginport, $beginfromsts,
$begintosts, $endslot, $endsubslot, $endport, $endfromsts, $endtosts];

			}
			$n++;
		}
		#print "$val\n" while defined($val = pop(@result));
	} elsif ($self->{'type'} eq "ONS15454") {
		$socket->print("RTRV-CRS:::$self->{'ctag'};\n");
		my $resp = $self->{'resp_queue'}->dequeue();
		my $n = 0;
		#print_resp($resp);
		while (defined($resp->[$n])) {
			# match "STS-6-1-73,STS-12-1-97:2WAY,STS24C:CKTID=\"CERN-Vancouver 2 - TST\":IS-NR,"
			if ($resp->[$n] =~ /"\w+-(\d+)-(\d+)-?(\d+)?,\w+-(\d+)-(\d+)-
?(\d+)?:[^,]+,([^:]+):CKTID=\\\"(.*)\\\":/) {
				my $id = $8;
				my $bandwidth = $7;
				my $beginslot = $1;
				my $beginsubslot = '';
				my $beginport = $2;
				my $beginfromsts = $3;
				my $endslot = $4;
				my $endsubslot = '';
				my $endport = $5;
				my $endfromsts = $6;
				my $speed;
				my $begintosts;
				my $endtosts;
				#calculate end sts, bandwidth is $7 from sts is $3 and $6
				if (defined $3){
					$speed = $bandwidth;
					$speed =~ s/\D//g;
					$begintosts = $beginfromsts + $speed - 1;
				}
				if (defined $6){
					$speed = $bandwidth;
					$speed =~ s/\D//g;
```

```
                                        $endtosts = $endfromsts + $speed - 1;
                                }
                                # catch undefined values
                                if (not defined $beginfromsts) {
                                        $beginfromsts = '';
                                }
                                if (not defined $begintosts) {
                                        $begintosts = '';
                                }
                                if (not defined $endfromsts) {
                                        $endfromsts = '';
                                }
                                if (not defined $endtosts) {
                                        $endtosts = '';
                                }

                                push @result, [$id, $speed, $beginslot, $beginsubslot, $beginport, $beginfromsts,
$begintosts, $endslot, $endsubslot, $endport, $endfromsts, $endtosts];
                        }
                        $n++;
                }

        }
        return @result;
    }



    #
    # Return (slot/port/sts) corresponding to argument (slot/port/sts)
    # This function accepts three arguments: slot, port, sts
    #
    sub get_xconnect_by_slotport {
        my ($self, $slot, $port, $sts) = @_;
        my $socket = $self->{'socket'};

        my $outslot;
        my $outport;
        my $outsts;
        my $swslot;
        my $swport;
        my $swsts;

        # try to find <(slot/port/sts),to> and possible switchmate match
        $socket->print("RTRV-CRS-STS3C::STS3C-1-$slot-$port-$sts,ALL:$self->{'ctag'}:::;\n");
        my $resp = $self->{'resp_queue'}->dequeue();
        my $n = 0;
        while (defined($resp->[$n])) {
                # match "STS3C-1-6-1-4,STS3C-1-1-4-1:"
                if ($resp->[$n] =~ /"STS3C-\d+-\d+-\d+-\d+,STS3C-\d+-(\d+)-(\d+)-(\d+):2WAYPR:SWMATE=STS3C-\d+-
(\d+)-(\d+)-(\d+):/) {
                        $outslot = $1;
                        $outport = $2;
                        $outsts = $3;
                        $swslot = $4;
                        $swport = $5;
                        $swsts = $6;
                } elsif ($resp->[$n] =~ /"STS3C-\d+-\d+-\d+-\d+,STS3C-\d+-(\d+)-(\d+)-(\d+):/) {
                        $outslot = $1;
                        $outport = $2;
                        $outsts = $3;
                }
                $n++;
        }

        # try to find <from,(slot/port/sts)> and possible switchmate match
        $socket->print("RTRV-CRS-STS3C::ALL,STS3C-1-$slot-$port-$sts:$self->{'ctag'}:::;\n");
        $resp = $self->{'resp_queue'}->dequeue();
        $n = 0;
        while (defined($resp->[$n])) {
                # match "STS3C-1-6-1-4,STS3C-1-1-4-1:"
                if ($resp->[$n] =~ /"STS3C-\d+-(\d+)-(\d+)-(\d+),STS3C-\d+-\d+-\d+-\d+:2WAYPR:SWMATE=STS3C-\d+-
(\d+)-(\d+)-(\d+):/) {
                        $outslot = $1;
                        $outport = $2;
                        $outsts = $3;
                        $swslot = $4;
                        $swport = $5;
                        $swsts = $6;
                } elsif ($resp->[$n] =~ /"STS3C-\d+-(\d+)-(\d+)-(\d+),STS3C-\d+-\d+-\d+-\d+:2WAYPR:SWMATE=STS3C-
\d+-(\d+)-(\d+)-(\d+):/) {
                        $outslot = $1;
                        $outport = $2;
                        $outsts = $3;
                }
                $n++;
        }
        return($outslot, $outport, $outsts, $swslot, $swport, $swsts);
    }


    #
    # Return circuit-id corresponding to argument (slot/port/sts)
# This function accepts three arguments: slot, port, sts
# It returns the label of the crossconnect.
#
sub get_cktid_by_slotport {
        my ($self, $slot, $port, $sts) = @_;
        my $socket = $self->{'socket'};

        $socket->print("RTRV-CRS-STS3C::STS3C-1-$slot-$port-$sts,ALL:$self->{'ctag'}:::DISPLAY=PROV,CKTID=ALL;\n");
        my $resp = $self->{'resp_queue'}->dequeue();
        my $n = 0;
        while (defined($resp->[$n])) {
                if ($resp->[$n] =~ /CKTID=\\"(.*)\\"/) {
                        return $1;
                }
                $n++;
        }
        $socket->print("RTRV-CRS-STS3C::ALL,STS3C-1-$slot-$port-$sts:$self->{'ctag'}:::DISPLAY=PROV,CKTID=ALL;\n");
        $resp = $self->{'resp_queue'}->dequeue();
        $n = 0;
        while (defined($resp->[$n])) {
                if ($resp->[$n] =~ /CKTID=\\"(.*)\\"/) {
                        return $1;
                }
                $n++;
        }
        return(undef());
}


#
# Return the "section trace" string received on $shelf/$slot/$port
# This function accepts two arguments: slot, port
#
sub get_section_trace {
        my ($self, $shelf, $slot, $port) = @_;

        my $speed;
        my $socket = $self->{'socket'};
        my $cardtype = get_cardtype($self, $shelf, $slot);
        if (!defined($cardtype)) {
                return undef();
        }
        if ($cardtype =~ /\d+xOC-(\d+)/) {
                $speed = $1;
        } else {
                return undef();
        }
        $socket->print("RTRV-TRC-OC${speed}::" .
                "OC$speed-$shelf-$slot-${port}:$self->{'ctag'}::INCSTRC;\n");
        my $resp = $self->{'resp_queue'}->dequeue();
        my $n = 0;
        while (defined($resp->[$n])) {
                if ($resp->[$n] =~ /"OC\d+-\d+-\d+-\d+:\\"(.*?)\\"/) {
                        my $trace = $1;
                        return $trace;
                }
                $n++;
        }
        return(undef());
}



#
# Return the Autodiscovery strings for $shelf/$slot/$port
# returns: (TX_TAG, RX_ACTUAL)
#
sub get_AD_by_slotport {
        my ($self, $shelf, $slot, $port) = @_;

        my $socket = $self->{'socket'};
        $socket->print("RTRV-AD-ALL:::$self->{'ctag'};\n");
        my $resp = $self->{'resp_queue'}->dequeue();
        my $n = 0;
        while (defined($resp->[$n])) {
                if ($resp->[$n] =~ /OC\d+-${shelf}-${slot}-${port}.*,TX_TAG=\\"(.*?)\\",.*?,RX_ACTUAL=\\"(.*?)\\"/) {
                        return ($1, $2);
                }
                $n++;
        }
        return(undef());
}



#
# Return the $shelf/$slot/$port belonging to AD string $id
# returns: ($shelf, $slot, $port)
#
sub get_AD_by_id {
        my ($self, $id) = @_;

        my $socket = $self->{'socket'};
        $socket->print("RTRV-AD-ALL:::$self->{'ctag'};\n");
        my $resp = $self->{'resp_queue'}->dequeue();
        my $n = 0;
        while (defined($resp->[$n])) {
                if ($resp->[$n] =~ /"OC\d+-(\d+)-(\d+)-(\d+).*,TX_TAG=\\"$id\\",/) {
                        return ($1, $2, $3);
                }
                $n++;
        }
        return(undef());
}


#
# Return IP address and name of NE with given ID
# Accept 1 argument: $id (12 hex numbers upper case - Ethernet address?)
# Return ($address, $name)
#
sub get_neighbour {
        my ($self, $id) = @_;

        $id =~ tr/a-z/A-Z/;
        my $socket = $self->{'socket'};
        $socket->print("RTRV-RTG-INFO:::$self->{'ctag'};\n");
        my $resp = $self->{'resp_queue'}->dequeue();
        my $n = 0;
        while (defined($resp->[$n])) {
                if ($resp->[$n] =~
                        /,\\"(.*?)\\",$id\|(\d+\.\d+\.\d+\.\d+),/) {
                        return ($2, $1);
                }
                $n++;
        }
}


#
# Three parameters: shelf, slot, port
# Return the Errored Seconds Section count for both Near and Far End
# and both Transmit and Receive
# Currently, it has the speed OC48 hardcoded!!
#
sub get_ess {
        my ($self, $shelf, $slot, $port) = @_;
        my $socket = $self->{'socket'};
        $self->verbose( 1, "retrieving ESS\n" );
        $socket->print("RTRV-PM-ALL::ALL:$self->{'ctag'}::ES-S,0-UP,ALL,ALL,1-UNT,,,,;\n");
        my $resp = $self->{'resp_queue'}->dequeue();
        my $n = 0;
        while (defined($resp->[$n])) {
                # match "OC48-1-4-1,OC48:ESS,0,ADJ,NEND,RCV,1-UNT,01-01,00-00,0"
                if ($resp->[$n] =~ /"OC\d+-1-${slot}-${port},OC\d+:ES-S,(\d+)/) {
                        return $1;
                }
                $n++;
        }
        return(undef());
}


#
# no arguments, returns a list of shelf-slot-port strings of all the
# OC48 ports
#
sub get_oc48_ports {
        my ($self) = @_;
        my $socket = $self->{'socket'};

        $self->verbose( 1, "retrieving OC48 ports\n" );
        $socket->print("RTRV-OC48::OC48-1-ALL:$self->{'ctag'}:::,;\n");
        my $resp = $self->{'resp_queue'}->dequeue();
        my $n = 0;
        my @result;
        while (defined($resp->[$n])) {
                # match "OC48-1-4-1::TMGREF=Y,DCC=Y,,SDTH=6,,,,, ...
                if ($resp->[$n] =~ /"OC48-(\d+-\d+-\d+):/) {
                        push(@result, $1);
                }
                $n++;
        }
        return(@result);
}


sub check_resp {
        my $resp = shift;
        my $n = 0;
        while (defined($resp->[$n])) {
                if ($resp->[$n] =~ /M\s+\w+\s+(\w+)/) {
                        #print "check:",$resp->[$n],":",$1,"\n";
                        return $1;
```

```
		}
		$n++;
	}
}


# function to read output in normal format
sub print_resp {
	my ( $self, $resp ) = @_;
	my $n = 0;
	while (defined($resp->[$n])) {
		print $n, " ",  $resp->[$n], "\n";
		$n++;
	}
}


# Send TL1 command and forget response or use code referent to process response
sub async_cmd {
	my ( $self, $cmd ) = @_;
	$self->{'acmd_queue'}->enqueue($cmd);
}


sub deviceError {
	my ($self,$autoMsg,$myMsg)=@_;
	$self->verbose( 2,"Connection: $self->{'connectionId'} -Autonomous Message:@$autoMsg $$myMsg!\n");
	1;
}

1;
```

pol_tl1.pm

```
package pol_tl1;

use strict;
use CGI::Cookie;
use tl1;

sub getHostname{
	my ($self,$site)=@_;
	my $hostname="0";
	if($site eq "Renci"){
```

```
                  $hostname="192.168.201.4";
         }
         elsif($site eq "UNC"){
              $hostname="192.168.203.4";
         }
         elsif($site eq "Duke"){
              $hostname="192.168.202.4";
         }
         elsif($site eq "NCSU"){
              $hostname="192.168.204.4";
         }
         return $hostname;
}

sub loginSwitch{

        my ($self,$connectionId,$hostname,$device,$port,$username,$password)=@_;

        my $logincmd="act-user::$username:1::$password;";

         #print STDERR "Login Hostname:", $hostname,"\n";

        my $tl1 = tl1->new(
                  $connectionId,
              hostname => $hostname,
              username => $username,
              password => $password,
              type=>$device,
              peerport => $port,
              verbose  => 0
        );

        # connect and login
        if ($tl1->open() == 0) {
              print STDERR &currentTime(),": Could not connect to $hostname Polatis\n";
              return undef;
        }
         print STDERR &currentTime(),": ID$connectionId-Logged into $hostname Polatis.\n";
        return $tl1;
}

#make CRS between 2 pairs of ports
sub makePairCRS{
         my ($self,$tl1,$site,$inputport1,$outputport1,$inputport2,$outputport2,$userName, $userPassword)=@_;

        my $resp=makeCRS($self,$tl1,$site,$inputport1,$outputport1, $userName, $userPassword);
        my $result=responseCheck($resp);

         return $result if($result ne "COMPLD");

         $resp=makeCRS($self,$tl1,$site,$inputport1,$outputport1, $userName, $userPassword);
        $result=responseCheck($resp);

        return $result;
    }

    sub makePairCRSOnly{
        my ($self,$tl1,$inputport1,$outputport1,$inputport2,$outputport2)=@_;

        my $resp=makeCRSOnly($self,$tl1,$inputport1,$outputport1);
        my $result=responseCheck($resp);

        return $result if($result ne "COMPLD");

        $resp=makeCRSOnly($self,$tl1,$inputport2,$outputport2);
        $result=responseCheck($resp);

        return $result;
    }


    # Make CRS between 2 ports
    sub makeCRS{
        my ($self,$tl1,$site,$inputport,$outputport,$username,$password)=@_;

        my $crscmd="ent-patch::$inputport,$outputport:1:;\n";

        my $resp=$tl1->cmd($crscmd);
        my $result=responseCheck($resp);

        print STDERR "CRS($inputport,$outputport) Result: ",$result,"\n";

        my $birthTime=current_date();   #Sth is wrong here
        #my $birthTime=undef;
        if($result eq "COMPLD"){
         my $crs_name=$inputport.'-'.$outputport;
            print STDERR $result,$username,$password;
            my $dbh = DBI->connect("DBI:mysql:database=ben;host=localhost",$username, $password,{'RaiseError' => 1})
or die("DBI connection fail \n");
            print STDERR $result,$site, $inputport, $outputport;
            $dbh->do("INSERT INTO crs VALUES (?, ?, ?, ?, ?,
?,?)",undef,$crs_name,$username,$site,$inputport,$outputport,$birthTime,undef) or die("DBI insert fail \n");
```

```
                $dbh->disconnect();
        }

        return $result;
}

# Make CRS between 2 ports
sub makeCRSOnly{
        my ($self,$tl1,$inputport,$outputport)=@_;

        my $crscmd="ent-patch::$inputport,$outputport:1:;\n";

        my $resp=$tl1->cmd($crscmd);
        my $result=responseCheck($resp);

        print STDERR "CRS($inputport,$outputport) Result: ",$result,"\n";

        return $result;
}

#To see it's "COMPLD", "DENIED"
sub responseCheck{
        my $resp=shift;

        my $result;

        my $n=0;
        while (defined($resp->[$n])) {
                if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
                        $result = $1;
                        print "ok:",$resp->[$n],":",$1,"\n";
                }
                print "result:",$result,"\n";
                $n++;
        }
        return $result;
}

sub retrievePort{
         my ($self,$tl1,$port)=@_;

        my $n=0;
        my $flag="FAIL";
        my ($status,$power,$outputport);
```

```
#retrieve port CRS
my $rtrvportcmd= "rtrv-patch::$port:1:;\n";
my $resp=$tl1->cmd($rtrvportcmd);
while (defined($resp->[$n])) {
    print STDERR "Retrieving port CRS $port:",$resp->[$n],"\n";

    if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
        $flag = $1;
        print STDERR "Retrieving port CRS $port:",$resp->[$n],":",$flag,"\n";
    }

    if($resp->[$n] =~ /\s*"(\d+),(\d+)/){
        if($port==$1){
                        $outputport=$2;
                }
                else{
                        $outputport=$1;
                }
    }
          $n++;
}

if($flag ne "COMPLD"){
    return("FAIL",undef,undef,undef);
}


 # retrieve port status
 $rtrvportcmd= "rtrv-port-shutter::$port:1:;\n";
$resp=$tl1->cmd($rtrvportcmd);
 $n=0;
while (defined($resp->[$n])) {
    if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
        $flag = $1;
        print STDERR "Retrieving port Status $port:",$resp->[$n],":",$flag,"\n";
    }

          if($resp->[$n] =~ /\s*"(\d+):(\w+)/){
                $status=$2;
          }
          $n++;
 }

if($flag ne "COMPLD"){
    return("FAIL",undef,undef,$outputport);
}
```

```
         #retrieve port power
        $rtrvportcmd="rtrv-port-power::$port:1:;\n";
        $resp=$tl1->cmd($rtrvportcmd);
        $n=0;
         while (defined($resp->[$n])) {
            if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
                $flag = $1;
                print STDERR "Retrieving port Power $port:",$resp->[$n],":",$flag,"\n";
            }

            if($resp->[$n] =~ /\d+:-(\d+)\.(\d+)/){
                          $power=(-1)*$1+(0.01)*$2;
            }
            $n++;
        }

        if($flag ne "COMPLD"){
            return("FAIL",$status,undef,$outputport);
        }

        return ("SUCCESS",$status,$power,$outputport);


    }

    sub dltCRS{
         my ($self,$tl1,$site,$inputport,$outputport,$userName, $password)=@_;

         my $dltcrscmd="dlt-patch::$inputport:1:;\n";
        my $resp=$tl1->cmd($dltcrscmd);
        my $n=0;
        my $flag;
         while (defined($resp->[$n])) {
            if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
                $flag = $1;
                print STDERR "Deleting CRS port: $inputport:",$resp->[$n],":",$flag,"\n";
            }
            $n++;
        }

        if($flag ne "COMPLD"){
            return("FAIL");
        }
```

```
                my $dbh = DBI->connect("DBI:mysql:database=ben;host=localhost",$userName, $password,{'RaiseError' => 1}) or
die "Cann't connect to Mysql\n";

                my $crs = $dbh->prepare("delete FROM crs where site=\'$site\' AND (port1=\'$inputport\' OR
port2=\'$inputport\')");
                $crs->execute();
                $dbh->disconnect();
                print STDERR "deleted from mysql\n";
                return ("SUCCESS");
        }

        sub dltCRSOnly{
                my ($self,$tl1,$site,$inputport,$outputport)=@_;

                my $dltcrscmd="dlt-patch::$inputport:1:;\n";
                my $resp=$tl1->cmd($dltcrscmd);
                my $n=0;
                my $flag;
                while (defined($resp->[$n])) {
                        if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
                                $flag = $1;
                                print STDERR "Deleting CRS port: $inputport:",$resp->[$n],":",$flag,"\n";
                        }
                        $n++;
                }

                if($flag ne "COMPLD"){
                        return("FAIL");
                }

                return("SUCCESS");
        }

        sub dltALL{
                my ($self,$tl1)=@_;
                my @interfaces=(1..32);
                my $dltcrscmd;
                my $resp;
                foreach my $if (@interfaces){
                                $dltcrscmd="dlt-patch::$if:1:;\n";
                                $resp=$tl1->cmd($dltcrscmd);
                                print $resp,":",$if,"\n";
                }
        }

        sub retrieveSwitch{
```

```
    my ($self,$hostname)=@_;
    my $tl1=loginSwitch($self,$hostname);

    my $rtrvpatchcmd="rtrv-patch:::1:;\n";
    my $resp=$tl1->cmd($rtrvpatchcmd);

    my ($result,@crs_array);
    my $n=0;
    while (defined($resp->[$n])) {
        if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
            $result = $1;
            print STDERR "Retrieving:",$resp->[$n],":",$1,"\n";
        }
        if($resp->[$n] =~ /\s*"(\d+),(\d+)/){
            #$result=$1."-".$2;
            $crs_array[$1]=$2;
            $crs_array[$2]=$1;
        }

        $n++;
    }

    $tl1->close();

    print "Retrieve Switch CRS Result: ",$result,"\n";

    return ($result, @crs_array);
}

sub retrieveCRS{

    my ($self,$hostname)=@_;
    my $tl1=loginSwitch($self,$hostname);

    my $rtrvpatchcmd="rtrv-patch:::1:;\n";
    my $resp=$tl1->cmd($rtrvpatchcmd);

    my ($result,@crs_array);
    my $n=0;
    while (defined($resp->[$n])) {
        if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
            $result = $1;
            #print STDERR "Retrieving:",$resp->[$n],":",$1,"\n";
        }
        if($resp->[$n] =~ /\s*"(\d+),(\d+)/){
                #$result=$1."-".$2;
                $crs_array[$1]=$2;
                #$crs_array[$2]=$1;
            }

            $n++;
        }

        $tl1->close();
        #print "Retrieve Switch CRS Result: ",$result,"\n";

        return ($result, @crs_array);
    }

    # Retrive crs from mysql database

    sub retrieveMysql{
        my ($class,$site, $userName, $userPassword) = @_;
        my $dbh = DBI->connect("DBI:mysql:database=ben;host=localhost",$userName, $userPassword,{'RaiseError' => 1})
or die "Cann't connect to Mysql\n";

        my $allcrs;
        my ($i,$j)=(0,0);
        my @port_color= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
        my ($port1,$port2);

        my $color_scheme=colors_16();
        my @color=$color_scheme->colors();
        my $crs = $dbh->prepare("SELECT * FROM crs where site='$site'");
        $crs->execute();
        while (my $ref = $crs->fetchrow_hashref()) {
            my @onecrs = ($ref->{'name'}, $ref->{'owner'}, $ref->{
    'site'}, $ref->{'port1'}, $ref->{'port2'}, $ref->{'birth'}, $ref->{'death'});
            $port1=$ref->{'port1'};
            $port2=$ref->{'port2'};
            $port_color[$port1-1]="#".$color[$i];
            $port_color[$port2-1]="#".$color[$i];
            $allcrs=$allcrs.join(",",@onecrs);
            $allcrs=$allcrs.";";
            #print $port1,";",$port2,";",$i,";",$port_color[$port1-1], ";", $port_color[$port2-1],"\n";
            $i++;
        }
        my $numCRS=$i;
        my $color_array=join(",",@port_color);
        #print $numCRS,$allcrs,$color_array,"\n";
        $crs->finish();
        $dbh->disconnect();

        return ($numCRS,$allcrs,$color_array);     #@allcrs address??
    }

    # Get the user priviledge via the session
    sub getSession(){
        my ($self,$crs)=@_;
        my $cookie=$crs->cookie(-name => "session");
        if($cookie) {
            CGI::Session->name($cookie);
        }
        my $session = new CGI::Session("driver:File",$cookie,{'Directory'=>"/tmp"}) or die "$!";
        my $priviledge = $session->param('priviledge');
        my $name = $session->param('name');
         my $password=$session->param('password');
        my $userName=$session->param('userName');
        return ($name,$userName,$priviledge,$password);
    }



    #print login portion of the homepage
    sub printLogin(){
        my ($self,$name,$priviledge)=@_;
        print "<div id=\"headerlogin\">";
        if( ($priviledge eq "Admin") | ($priviledge eq "Provisioning")){
            print "<span class=\"h2\"> Welcome!, $name ($priviledge) </span>";
                  print "<br>";
            print  "<form action=\"./login.cgi\" method=\"post\">";
            print "<input type=submit value=logout> &nbsp&nbsp";
            print "</form>";
        }
        else{
            print "<form action=\"./login.cgi\" method=\"post\">";
            print "Yourname: <input name=name id=name type=text size=25>";
            print "   Password: <input name=pwd id=pwd type=password size=25> <br><br>"
    ;
            print "<input type=submit value=login> &nbsp&nbsp";
            print "<input type=reset value=Reset name=reset><br>";
            print "</form>";
        }
        print "</div>";
    }

    #Print ilegal warning
    sub printWarning (){
```

```
            my ($self,$name)=@_;
            print  "<div id=\'content\'>";
            print "<span class=\"h3\"> Hello! $name,You need to have the Admin or Provisioning priviledge to provision
crossconnects. Please try following: </span><br>";
            print "<span class=\"h3\">Retry the login, you current session may expire.Or </span><br>";
            print "<span class=\"h3\">Contact: yxin\@renci.org to get an account </span><br>";
            print "</div>";
      }



      #Return current local date
      sub current_date{

            my ($sec,$min,$hour,$mday,$month,$year,$wday)=localtime();
             #(
            #      localtime->sec(),
            #      localtime->min(),
            #      localtime->hour(),
            #      localtime->wday(),
            #      localtime->mon()+1,
            #      localtime->year(),
            #      localtime->mday()
            #);
            $month="0".$month if ($month)<10;
            #make month in 2 digits if it is less than 2

            $mday="0".$mday if ($wday)<10;
            #make date in 2 digits if it is less than 2

            my $yr=substr($year,1);
            $yr="20".$yr if length($yr)==2;
            #make year in 4 digits

            my $date=$yr."-".$month."-".$mday;
            #concatenate the date in yyyy-mm-dd format.

            return $date;
      }

      sub colors_16{
         my $scheme = Color::Scheme->new
            ->from_hex('ff0000') # or ->from_hue(0)
            ->scheme('tetrade')
            ->distance(0.3)
            ->add_complement(0)
```

```
        ->variation('pastel')
        ->web_safe(1);

    return $scheme;
}

sub find_power{

        my ($self,$tl1,$port)=@_;

        my $n=0;
        my $flag="FAIL";
        my $power;

        #retrieve port power
        my $rtrvportcmd="rtrv-port-power::$port:1:;\n";
        my $resp=$tl1->cmd($rtrvportcmd);
        $n=0;
        while (defined($resp->[$n])) {
              if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
                     $flag = $1;
                                    #print STDERR "Retrieving port Power $port:",$resp->[$n],":",$flag,"\n";
              }

              if($resp->[$n] =~ /\d+:-(\d+)\.(\d+)/){
                                    $power=(-1)*$1+(0.01)*$2;
              }
              $n++;
        }

        if($flag ne "COMPLD"){
              return("FAIL",undef);
        }

        return ("SUCCESS",$power);

        }



sub logoutSwitch{
        my ($self,$tl1)=@_;
        $tl1->close();
        print STDERR &currentTime(),": ID$tl1->{'connectionId'}-Logged out from $tl1->{hostname} Polatis.\n";
        }

#returns the current date and timestamp in the format yyyy-mm-dd hh24:mi:ss
```

```
sub currentTime{
    my $current_time;
    my ($sec,$min,$hour,$mday,$mon,$year,$wday,
    $yday,$isdst)=localtime(time);
    $current_time=sprintf("%4d-%02d-%02d %02d:%02d:%02d",
    $year+1900,$mon+1,$mday,$hour,$min,$sec);
    return $current_time;
    }



1;
```

infinera_tl1.pm

```
package infinera_tl1;

use strict;
use tl1;

#Returns the IP Address of the host
sub getHostname{
     my ($self,$site)=@_;
     my $hostname="0";
     if($site eq "Renci"){
                $hostname="192.168.201.5";
     }
     elsif($site eq "UNC"){
          $hostname="192.168.203.5";
    }
     elsif($site eq "Duke"){
          $hostname="192.168.202.5";
    }
     elsif($site eq "NCSU"){
          $hostname="192.168.204.5";
     }
     return $hostname;
}

#Creates connection object & logs in
sub loginSwitch{

    my ($self,$connectionId,$hostname,$device,$port,$username,$password)=@_;

    my $logincmd="act-user::$username:1::$password;";
```

```
        my $tl1 = tl1->new(
                $connectionId,
            hostname => $hostname,
            username => $username,
            password => $password,
            type=>$device,
            peerport => $port,
            verbose  => 0
        );

        # connect and login
        if ($tl1->open() == 0) {
                print STDERR &currentTime(),": Unable to connect to $hostname\n";
    =begin comment #uncomment if this happens too often
            print STDERR &currentTime(),": First attempt to connect to $hostname failed. Trying again...\n";
                sleep 1;
                if($tl1->open()==0){
                        print STDERR &currentTime(),": Second(Final) attempt to connect to $hostname Infinera failed.
Abort.\n";
                        return undef;
                        }
    =end comment
    =cut
                return undef;
        }
        print STDERR &currentTime(),": ID$connectionId-Logged into $hostname Infinera.\n";
        return $tl1;
    }

    #To see status of response: "COMPLD", "DENIED"
    sub responseCheck{
        my $resp=shift;

        my $result;

        my $n=0;
        while (defined($resp->[$n])) {
            if($resp->[$n] =~ /^M\s+\w+\s+(\w+)/){
                $result = $1;
            }
            $n++;
        }
        return $result;
    }

    #Fires command for power measurement and returns the power value in dB
    sub findPower{
        my ($self,$tl1,$chassis,$DLMslot,$opticalChannel)=@_;

        my $pwrCmd='rtrv-pm-och::'.$chassis.'-A-'.$DLMslot.'-L1-'.$opticalChannel.':rxpwr::opr;';

        my $pwrResp=$tl1->cmd($pwrCmd);

        unless(defined($pwrResp)){
                print STDERR &currentTime(),":ID$tl1->{'connectionId'}- Command $pwrCmd did not get any response!
User Logged Out! Infinera\n";
                return "COMMAND $pwrCmd Failed to get any response!";
                }

        my $result=responseCheck($pwrResp);

        if($result ne "COMPLD"){
                return "COMMAND $pwrCmd FAILED! RESPONSE:@$pwrResp";
                }

        if ($pwrResp->[2]=~/"$chassis-.*?[:,].*?[,:].*?[,:]\s*?([-+]?[0-9.]*?)\s*?,/){
                return $1;
                }
        else{
                return "INVALID RESPONSE:@$pwrResp";
                }

        }

    #Fires the command for BER measurement and returns the BER
    sub findBER{

        my ($self,$tl1,$chassis,$DLMslot,$opticalChannel,$corFlag)=@_;

        my $monType;
        ($corFlag==0) ? ($monType="BERPREFEC") : ($monType="BERPOSTFEC");

        my $berCmd='rtrv-pm-och::'.$chassis.'-A-'.$DLMslot.'-L1-'.$opticalChannel.':rxber::'.$monType.';';

        my $berResp=$tl1->cmd($berCmd);

        unless(defined($berResp)){
                print STDERR &currentTime(),":ID$tl1->{'connectionId'}- Command $berCmd did not get any response!
User Logged Out! Infinera\n";
                return "COMMAND $berCmd Failed to get any response!";
                }
        my $berResult=responseCheck($berResp);

        if($berResult ne "COMPLD"){
                return "COMMAND $berCmd FAILED!  RESPONSE:@$berResp";
                }

        if ($berResp->[2]=~/"$chassis-.*?[:,].*?[:,].*?[:,]\s*?([0-9.E-]+)\s*?,/){
                return $1;
                }
        else{
                return "INVALID RESPONSE FOR COMMAND $berCmd!  RESPONSE:@$berResp";
                }

        }

    #Logs out from Infinera
    sub logoutSwitch{
        my ($self,$tl1)=@_;
        $tl1->close();
        print STDERR &currentTime(),":ID$tl1->{'connectionId'}-Logged out from $tl1->{hostname} Infinera.\n";
        }

    #returns the current date and timestamp in the format yyyy-mm-dd hh24:mi:ss
    sub currentTime{
        my $current_time;
        my ($sec,$min,$hour,$mday,$mon,$year,$wday,
        $yday,$isdst)=localtime(time);
        $current_time=sprintf("%4d-%02d-%02d %02d:%02d:%02d",
        $year+1900,$mon+1,$mday,$hour,$min,$sec);
        return $current_time;
        }

    1;
```

6.2 Perl Scripts

XMLRPC_API_Server_MH_InfineraPolatis.pl

```
    use infinera_tl1;
    use pol_tl1;
    use XMLRPC::Transport::HTTP;
    use Switch;
    use strict;
    use warnings;
    #use XMLRPC::Lite +trace => 'debug';#uncomment to check XML requests and resp
    use threads;
    use threads::shared;
    use Thread qw(async);
    use Thread::Semaphore;

    my $daemon = XMLRPC::Transport::HTTP::Daemon
                    ->new(LocalPort => '8001')
                    ->dispatch_to('connectToDevice','retrieveMeasurement','disconnectFromDevice');

    my $mutex=Thread::Semaphore->new();
    my %connections;
    my %connectionTime :shared ;
    my $connectionId = 1000; #starting connection Id=1001
    my $timeout :shared =5*60; #timeout after 5 minutes. CAUTION: Do not exceed device timeout(12minutes)

    sub connectToDevice{

        shift;
        my $ConParam=shift;

        my $hostname = $ConParam->{'deviceIP'} or return {ERROR=>"Value for deviceIP not present in parameter"};
        my $port = $ConParam->{'port'} or return {ERROR=>"Value for port not present in parameter"};
        my $username = $ConParam->{'username'} or return {ERROR=>"Value for username not present in parameter"};
        my $password = $ConParam->{'password'} or return {ERROR=>"Value for password not present in parameter"};
        my $device = $ConParam->{'device'} or return {ERROR=>"Value for device not present in parameter"};

        $mutex->down();

        if(uc $device eq 'POLATIS'){
                    $connectionId++;
                    $connections{$connectionId}=pol_tl1-
>loginSwitch($connectionId,$hostname,$device,$port,$username,$password);
                    }
        elsif (uc $device eq 'INFINERA'){
                    $connectionId++;
                    $connections{$connectionId}=infinera_tl1-
>loginSwitch($connectionId,$hostname,$device,$port,$username,$password);
                    }
        else{
                    $mutex->up();
                    return {ERROR=>"Invalid Device! Try Polatis or Infinera!"};
                    }

        if(defined($connections{$connectionId})){
                    $connectionTime{$connectionId}=time;
```

```
                my $conObject=$connectionId;
                $mutex->up();

                my $timeout_thread= async{ #thread for timeout
                        sleep($timeout);
                        while($connectionTime{$conObject}){
                                if($connectionTime{$conObject} <= time-$timeout){
                                &logoff($conObject);
                                        return 1;
                                        }
                                else{
                                        my $nowtime=time;
                                        my $timeToSleep=$timeout-($nowtime-$connectionTime{$conObject});
                                        sleep($timeToSleep);
                                        }
                                }
                        };
                $timeout_thread->detach();

                return {connectionObject=>$connectionId};
                }
        else{
                delete($connections{$connectionId--});
                $mutex->up();
                return {'ERROR'=>"Failed to connect to $hostname;"};
                }

        }

    sub retrieveMeasurement{

        shift;
        my $param=shift;
        my $monType;

        my $conObject = $param->{'connectionObject'} or return {'ERROR'=>"ConnectionObject not passed as
parameter!"};
        $mutex->down();

        unless(defined($connectionTime{$conObject})){
                delete $connections{$conObject} if(defined($connections{$conObject}));
                $mutex->up();
                my $timeoutMin=$timeout/60;
                return {ERROR=>"connectionObject $conObject not active. Connections get timed out after $timeoutMin
minutes idle time!"};
                }
        $connectionTime{$conObject}=time;

        if(uc $connections{$conObject}->{type} eq 'INFINERA'){

                my $chassis;
                if(defined($param->{'chassis'})){
                        $chassis = $param->{'chassis'};
                        }
                else{
                        $mutex->up();
                        return {'ERROR'=>"Chassis name not passed as parameter!"};
                        }

                my $DLMslot;
                if(defined($param->{'DLMslot'})){
                        $DLMslot=$param->{'DLMslot'};
                        }
                else{
                        $mutex->up();
                        return {'ERROR'=>"DLMslot name not passed as parameter!"};
                        }

                my $opticalChannel;
                if(defined($param->{'opticalChannel'})){
                        $opticalChannel=$param->{'opticalChannel'};
                        }
                else{
                        $mutex->up();
                        return {'ERROR'=>"opticalChannel name not passed as parameter!"};
                        }

                if(defined($param->{'measureType'})){
                        $monType=$param->{'measureType'};
                        }
                else{
                        $mutex->up();
                        return {'ERROR'=>"MeasureType not passed as parameter!"};
                        }


                switch (uc $monType){
                        case "PORTPOWER"{

                                my $power=infinera_tl1-
>findPower($connections{$conObject},$chassis,$DLMslot,$opticalChannel);
```

```
                                        if($power=~/^[0-9-.]*$/){
                                                $mutex->up();
                                                return {'POWER'=>$power};
                                                }
                                        else{
                                                if($power=~/logged out/i){
                                                        delete($connections{$conObject});
                                                        delete($connectionTime{$conObject});
                                                        }
                                                $mutex->up();
                                                return {'ERROR'=>$power};
                                                }

                                        }

                                case "PREFEC-BER"{

                                        my $unCorrBER=infinera_tl1-
>findBER($connections{$conObject},$chassis,$DLMslot,$opticalChannel,0);

                                        if($unCorrBER =~ /^[0-9.E-]*$/){
                                                $mutex->up();
                                                return {'preFEC-BER'=>$unCorrBER};
                                                }
                                        else{
                                                if($unCorrBER=~/logged out/i){
                                                        delete($connections{$conObject});
                                                        delete($connectionTime{$conObject});
                                                        }
                                                $mutex->up();
                                                return {'ERROR'=>$unCorrBER};
                                                }

                                        }

                                case "POSTFEC-BER"{

                                        my $corrBER=infinera_tl1-
>findBER($connections{$conObject},$chassis,$DLMslot,$opticalChannel,1);

                                        if($corrBER =~ /^[0-9.E-]*$/){
                                                $mutex->up();
                                                return {'postFEC-BER'=>$corrBER};
                                                }
                                        else{
```

```
                                        if($corrBER=~/logged out/i){
                                                delete($connections{$conObject});
                                                delete($connectionTime{$conObject});
                                                }
                                        $mutex->up();
                                        return {'ERROR'=>$corrBER};
                                        }

                                }

                        else {
                                $mutex->up();
                                return {'ERROR'=>"$monType not defined"};
                                }
                        }
                }
        else{ #POLATIS
                my $port;
                if(defined($param->{'port'})){
                        $port= $param->{'port'}
                        }
                else{
                        $mutex->up();
                        return {'ERROR'=>"Port name not passed as parameter!"};
                        }

                if(defined($param->{'measureType'})){
                        $monType = $param->{'measureType'};
                        }
                else{
                        $mutex->up();
                        return {'ERROR'=>"MeasureType not passed as parameter!"};
                        }

                switch (uc $monType){
                        case "PORTPOWER"{

                                my ($flag,$power)=pol_tl1->find_power($connections{$conObject},$port);

                                unless(defined($power)){
                                        $mutex->up();
                                        return {'ERROR'=>'Null value fetched'};
                                        }

                                if($flag eq "SUCCESS"){
                                        $mutex->up();
                                        return {'POWER'=>$power};
                                        }
                                else{
                                        if($power=~/logged out/i){
                                                delete($connections{$conObject});
                                                delete($connectionTime{$conObject});
                                                }
                                        $mutex->up();
                                        return {'ERROR'=>$power};
                                        }

                                }
                        else {
                                $mutex->up();
                                return {'ERROR'=>"$monType not defined"};
                                }
                        }

                }
        }

    sub disconnectFromDevice{
        shift;
        my $ConParam=shift;

        my $conObject=$ConParam->{connectionObject} or return {'ERROR'=>"connectionObject name not passed as
parameter!"};
        my $hostname = $ConParam->{'deviceIP'} or return {ERROR=>"Value for deviceIP not present in parameter"};
        my $port = $ConParam->{'port'} or return {ERROR=>"Value for port not present in parameter"};
        my $username = $ConParam->{'username'} or return {ERROR=>"Value for username not present in parameter"};
        my $password = $ConParam->{'password'} or return {ERROR=>"Value for password not present in parameter"};
        my $device = $ConParam->{'device'} or return {ERROR=>"Value for device not present in parameter"};


        if(defined($connectionTime{$conObject})){

                my $tl1=$connections{$conObject};

                return {'ERROR'=>"Incorrect username"} unless($username eq $tl1->{username});
                return {'ERROR'=>"Incorrect password"} unless($password eq $tl1->{password});
                return {'ERROR'=>"Incorrect deviceIP"} unless($hostname eq $tl1->{hostname});
                return {'ERROR'=>"Incorrect port"} unless($port eq $tl1->{peerport});
                return {'ERROR'=>"Incorrect device"} unless ((uc $device) eq (uc $tl1->{type}));

                &logoff($conObject);
                $mutex->down();
```

```
                        delete $connections{$conObject};
                        $mutex->up();
                        return {Disconnected=>$conObject};
                        }
            else{
                        if(defined($connections{$conObject})){
                                    $mutex->down();
                                    delete $connections{$conObject};
                                    $mutex->up();
                                    }
                        my $timeoutMin=$timeout/60;
                        return {ERROR=>"connectionObject $conObject not active. Connections get timed out after $timeoutMin
minutes idle time!"};
                        }
            }

      sub logoff{

            my ($conObject)=@_;
            my $tl1 = $connections{$conObject};
            my $device = $tl1->{type};

            $mutex->down();
            (uc $device eq 'INFINERA') ? (infinera_tl1->logoutSwitch($tl1)) : (pol_tl1->logoutSwitch($tl1));
            delete($connectionTime{$conObject});
            $mutex->up();
            }

      print "Contact to XMLRPC server at ",$daemon->url,"\n";
      eval{ $daemon->handle }; warn $@ if $@;

      __END__
```