Bare Metal Provisioning 3.0

Automating the Network

Overview, Implementation & Use Cases

Dell Technical Marketing – Data Center Networking
February 2013

This document is for informational purposes only and may contain typographical errors. The content is provided as is, without express or implied warranties of any kind.

February 2013 | Rev 1.0

Contents

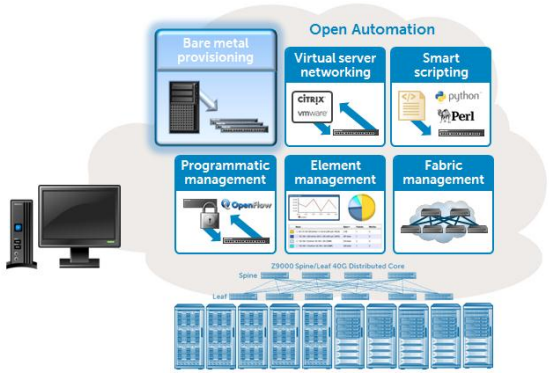
Figures

Overview

In today's fluid, automated, and highly reliable enterprise and data center environments, the ability to quickly provision network devices without user intervention has become paramount. Virtualization and cloud computing have been key drivers to rethinking network architecture and automating tasks that only a few years back would be time consuming and laborious. No longer is it required for a network engineer to go from switch to switch entering redundant commands or manually configuring each switch to meet network requirements.

With Open Automation, Dell Networking provides a solution for network provisioning that extends far beyond the basic requirements of network device configuration.

Imagine being able to simply rack, connect, and plug-in a switch, and have the switch take care of everything else. Imagine the switch self-testing all the basic connectivity requirements before downloading and applying the latest operating system image and correct configuration from a server. Further, imagine the switch self-testing the connectivity and functionality before sending the log file to an FTP server so the network engineer can know of the deployment results. This is the power of Bare Metal Provisioning 3.0 (BMP) that is part of Dell's Open Automation Framework.



Requirements & BMP Process

BMP is embedded in the latest FTOS image on every Dell S55, S60, S4810, S4820T, Z9000, and MXL blade switch, and it promises to save time, decrease downtime, cut costs, and provide reliability. BMP 3.0, which includes the additional functionality to download a script file instead of just a config file as in prior versions, is currently only available on the S4810 and Z9000 starting in FTOS 9.1.

Warranted by the trend and demand in the industry for automated services, BMP is the default boot mode on all supported Dell switches running the latest FTOS image. This means the following will occur when the switch boots-up:

1. When the switch boots up in BMP mode, all ports, including management ports, are placed in L3 mode in a 'no shut' state. The switch acts as a DHCP client on these ports for a period of time (dhcp-timeout). This allows the switch time to send out a DHCP DISCOVER on all 'up' interfaces to the DHCP Server in order to obtain its IP address, boot image file, and configuration/script file from the DHCP server. A DHCP request will be sent out on all ports.

2. The port that receives the first DHCP offer with a valid payload that contains the FTOS image info and config/script file path info will remain up and acquire the respective IP address based on the MAC address match in the 'dhcpd.conf" file. The other ports will be shut down.

3. The respective FTOS image will be downloaded and installed if it is different than what is already on the switch, and the respective config/script file will be downloaded and applied. **Note, even if the FTOS image is an older image, it will still be applied since it's different than what is installed on the switch.**

4. The IP address will be released back into the DHCP pool.

BMP is able to accomplish its tasks while taking careful security provisions. The scripts employed via BMP are run by a special system user, 'bmpuser', which cannot be used outside of the BMP process.

For BMP to work, a DHCP server must be setup and properly configured with the relevant information to respond to DHCP requests via the BMP process. Additionally, a FTP server needs to be setup to store the FTOS image and configuration or script files(s) used by BMP; other protocols such as TFTP, SFTP, HTTP, and HTTPS are also supported. Instructions on how to setup DHCP and FTP are provided in Appendix A and Appendix B.

Why & When to Use BMP

BMP improves accessibility to the switch by automatically loading pre-defined configurations and boot images that are stored on a file server. Additionally, new in BMP 3.0, scripts can be downloaded upon switch boot-up so that automated testing and configuration can be performed.

Some important drivers pushing the use of technologies like BMP are:

- *Simplified and fast deployment* – simply storing all the information to configure multiple switches on a centralized server and having switches pull the information and automatically test for prerequisites and configure themselves drastically cuts down on deployment and down-time.
- *Prevent errors* – no longer is it necessary to worry about end-user configuration errors; the configuration is written and tested once and stored on the DHCP server for consistently applying the same known working configuration when needed.
- *Decreased IT management cost* – this benefit directly correlates to the decreased overhead needed to manage switches. The configuration or script only needs to be written once and saved on the DHCP server. The same configuration can easily be applied to multiple switches or different switches may be set to download different configuration or script files. The more efficient management and less overhead directly correlate to cost savings.

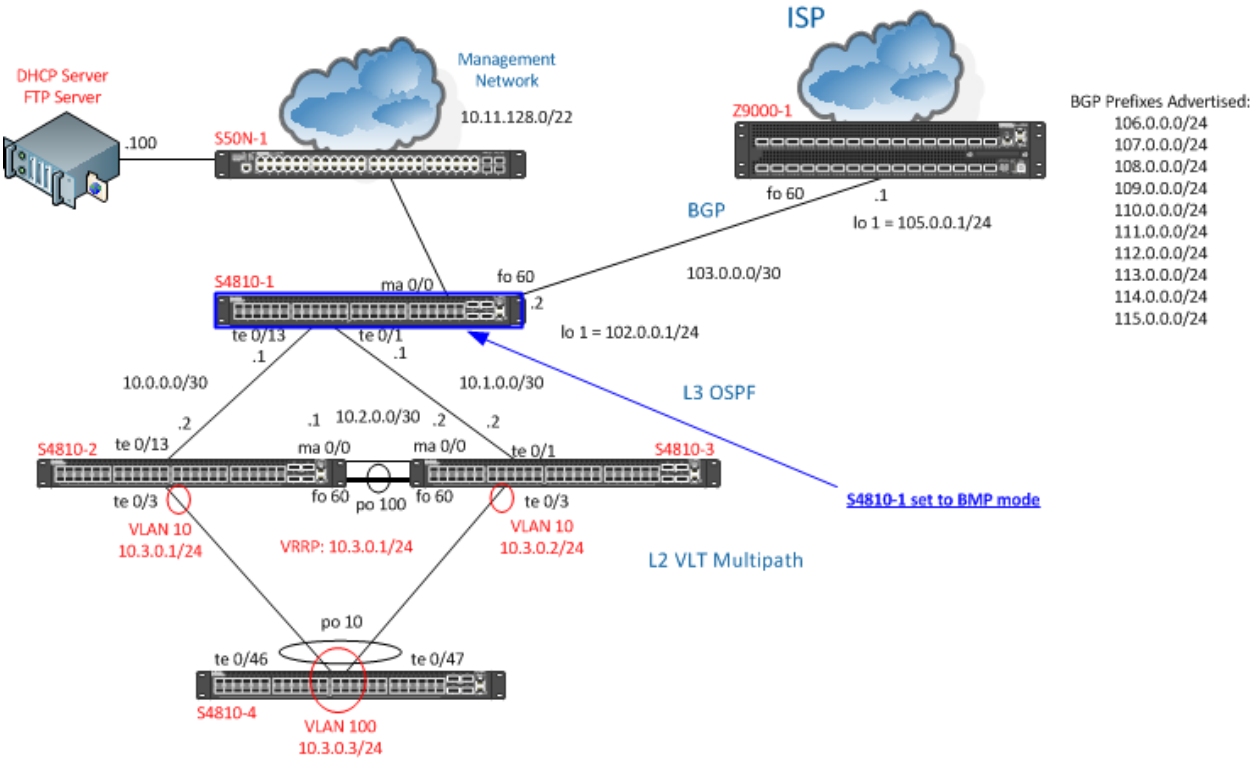
BMP is especially desirable for virtualized data centers where automation is used heavily. It assists in automating the network by providing the ability for scripts to run automatically upon switch boot-up; this provides automated testing and configuration capabilities that ensure reliability and minimal downtime. With the ability to download a script upon boot-up, a switch can make an intelligent decision on the appropriate action or configuration to apply based on the status of the surrounding environment.

Use Cases & Code

For all use cases in demonstrating BMP we will use the lab diagram below that is restricted to a small subsection of a possibly larger network design such as a spine and leaf architecture. The lab setup consists of four main parts:

- Management Network
- L2 LAN
- L3 core which may be part of a larger spine and leaf network
- BGP internet connection to the ISP

Figure 1: BMP Lab Setup



As you can see from the above diagram, switch 'S4810-1' is the switch in our network that will be used to demonstrate BMP. The surrounding switches in the network are already configured respectively with LLDP, VLT, OSPF, and BGP. In this lab we will demonstrate auto-configuration of switch 'S4810-1' using BMP. The three use cases that will be demonstrated in detail are:

1. Check for the latest FTOS image and download and apply configuration file
2. Check for the latest FTOS image and download and run script file for automated pre-requisite checks and, if pre-requisites are met, apply configuration via scripting
3. Check for the latest FTOS image and download and run script file for automated pre-requisite checks and, if pre-requisites are met, download and apply the full configuration file. Lastly, download and run a post-configuration script file to confirm successful configuration and status.

For the most up-to-date code related to the scripts in this whitepaper, download the file titled "BMP_3_0_Whitepaper_Scripts.zip" from the Dell TechCenter script site located at:

http://en.community.dell.com/techcenter/networking/m/force10_networking_scripts/default.aspx

Use Case 1: Check for the latest FTOS image and download and apply configuration file

This will be our simplest use case in which we check the DHCP server for the latest FTOS image and download the configuration file for the switch. Note that the FTOS image will only be downloaded if it is a different image than the one already installed on the switch. For detailed instructions on how to install a DHCP and FTP server to use with BMP see "Appendix A: Installing a DHCP Server" and "Appendix B: Installing a FTP Server".

Below is the content of the 'dhcpd.conf' file on our CentOS Linux server. The only section you will need to change in this file for your environment is the section for the range of IP addresses for the DHCP server to use and the specific host information section. Note, below, we only have one host section starting with 'host S4810-1' listed as for this example we're only using BMP with one switch. However, if you had three switches using BMP, you would simply copy and paste this host section two more times and update the respective fields. We provided a hostname of "host S4810-1" for reference, but this information is not used by BMP for anything so you can name it as you wish for reference purposes.

The 'hardware Ethernet' field is one of the most important fields in the host section as this is how the DHCP server will know what image file and config/script file to use with what switch. The 'hardware Ethernet' field should be set to the MAC address of the switch interface for the switch that will use the respective FTOS image and config/script file listed. Note, all interfaces have the same MAC address.

Also, in the host section, the 'filename' field should be set to the FTOS image path and the 'option configfile' field should be set to the config/script file path. In this use case we are simply downloading a config file, but in the next use case we will update the 'option configfile' field to point to a script file. The 'fixed-address' field needs to be set to the IP address we want the switch to receive and use to communicate with the DHCP and FTP server.

Code Snippet 1: 'dhcp.conf' from CentOS Linux server

```
#dhcp.conf file

ddns-update-style none;
ignore client-updates;

option configfile code 209 = text;

default-lease-time 100000;   # 24 hours
max-lease-time 200000;       # 48 hours


#range of IP addresses for DHCP server use and additional options
subnet 10.11.128.0 netmask 255.255.252.0
{
   range 10.11.129.1 10.11.129.253;
   option domain-name-servers 10.11.127.1;
}


#the host to use the below values for will be matched via interface mac-address
host S4810-1
{
   #interface mac-address of switch that matches this host
   hardware ethernet 00:01:e8:8a:de:50;

   #ip address that DHCP server will assign host that has above interface mac-address
   fixed-address 10.11.129.215;

   #FTOS image path to check for latest FTOS image
   filename "ftp://labadmin:password@10.11.129.100/FTOS_Image.bin";

   #configuration file or script file to be applied to the host
   option configfile "ftp://labadmin:password@10.11.129.100/scripts/S4810-1.conf";
}
```

As you can see from the above, the switch with the interface mac-address of (00:01:e8:8a:de:50) is set to receive the '10.11.129.215' IP address once the DHCP request from the switch is received. Further,

you can see that the DHCP server will check the image named 'FTOS_Image.bin' to see if it is different than the image already installed on the switch. If it is, the image will be downloaded and installed to the switch making the request and matching the respective mac-address. Also, note that the DHCP server is set to send the configuration file titled 'S4810-1.conf' to the switch.

Before reloading the switch we need to confirm that the FTP server is running and has the respective files in place. Additionally, we need to confirm that the switch is set to BMP mode. Checking the BMP mode can be accomplished via the 'show reload-type' command from Exec Privilege mode or 'enable' mode as shown below.

Figure 2: Confirm 'reload-type' is set to 'BMP' mode

```
S4810-1#show reload-type
Reload-Type             :   bmp [Next boot : bmp]
config-scr-download     :   enable
dhcp-timeout            :   2
user-defined-string     :
```

If needed, we can change the 'reload-type' to 'BMP' mode as shown below in configuration mode.

Figure 3: Change the 'reload-type' to BMP

```
S4810-1#conf
S4810-1(conf)#reload-type bmp config-scr-download ?
disable                 Disable the configfile download from the server
enable                  Enable the configfile download from the server
S4810-1(conf)#reload-type bmp config-scr-download enable
S4810-1(conf)#do wr
```

Notice from the above, if desired, we can disable the option to download the configuration file. This is important to remember and can come in handy if we make changes after the configuration file is initially downloaded. We need to disable the configuration file download to prevent future switch reloads from pulling the initial configuration file down again and overwriting any changes we made. If desired, we can also completely disable BMP by setting the 'reload-type' to 'normal' instead of 'bmp'.

Now that we confirmed all is good, we can go ahead and reload the switch with the 'reload' command from Exec Privilege/'enable' mode. As the switch reloads and starts to come back up, the switch will send out a DHCP request on all ports, and the DHCP server will assign the respective IP address based on the mac-address lookup it performs. In our case, the image is the same so no image will be downloaded; however, the configuration file needed for our lab setup shown in Figure 1 will be downloaded and applied.

As the switch loads and downloads the config file, the BMP process output will be seen as shown below. Notice, the switch does not hold on to the IP address given to it by the DHCP server; it simply uses the IP address to download the needed files and then releases it. However, in our case, the configuration file assigns the same IP address to the management interface.

Figure 4: BMP in process

```
00:02:06: %STKUNIT0-M:CP %JUMPSTART-5-JUMPSTART_DWNLD_CONFIG_SCRIPT_SUCCESS: The config/script file download is su
ccessful.
00:02:07: %STKUNIT0-M:CP %JUMPSTART-5-JUMPSTART_RELEASE: DHCP RELEASE sent on Ma 0/0.
00:02:07: %STKUNIT0-M:CP %JUMPSTART-5-JUMPSTART_DOWNLOAD: The config file download is successful.
00:02:07: %STKUNIT0-M:CP %JUMPSTART-5-CFG_APPLY: The downloaded config from dhcp server is being applied
00:02:07: %STKUNIT0-M:CP %JUMPSTART-2-JUMPSTART_DOWNLOAD_START: The config file download has started.
00:02:07: %STKUNIT0-M:CP %IFMGR-5-ASTATE_DN: Changed interface Admin state to down: Ma 0/0
00:02:08: %STKUNIT0-M:CP %IFMGR-5-OSTATE_DN: Changed interface state to down: Ma 0/0
00:02:08: %STKUNIT0-M:CP %SYS-5-CONFIG_LOAD: Loading configuration file
```

If we want to stop BMP after the switch has already reloaded in BMP mode, we can use the "stop bmp" command from Exec Privilege/'enable' mode. We can do this even after a script has started executing.

Use Case 1 demonstrated a simple but extremely powerful method for automating the configuration of multiple switches. Rather than manually applying configurations to each switch, simply load all the configs on a DHCP server, and, when needed, plug in the switch in its respective place in the network, and let BMP take care of the rest.

We know that this switch is in a crucial location in the network. We also know what OSPF and BGP neighbors it should have and that it should be able to communicate to nodes across the VLT link employed by its downstream switches. We can manually check this on the switch as shown below; however, in one of our later use cases we're going to use BMP to run a script that automates tasks like this for us.

Figure 5: Manually confirm BGP and OSPF neighbors and communication across downstream VLT

```
S4810-1#show ip bgp summary
BGP router identifier 102.0.0.1, local AS number 5
BGP table version is 11, main routing table version 11
11 network entrie(s) using 2050 bytes of memory
11 paths using 1056 bytes of memory
BGP-RIB over all using 1067 bytes of memory
1 BGP path attribute entrie(s) using 72 bytes of memory
1 BGP AS-PATH entrie(s) using 47 bytes of memory
1 neighbor(s) using 8192 bytes of memory

Neighbor        AS            MsgRcvd  MsgSent     TblVer  InQ  OutQ Up/Down  State/Pfx

104.0.0.1       10                 26       25         11    0     0 00:22:26        11
S4810-1#
S4810-1#show ip ospf neighbor

Neighbor ID     Pri      State         Dead Time Address         Interface Area
2.0.0.0          1       FULL/BDR      00:00:36 10.0.0.2         Te 0/13   0
3.0.0.0          1       FULL/BDR      00:00:35 10.1.0.2         Te 0/1    0
S4810-1#
S4810-1#ping 10.3.0.3

Type Ctrl-C to abort.

Sending 5, 100-byte ICMP Echos to 10.3.0.3, timeout is 2 seconds:
!!!!!
Success rate is 100.0 percent (5/5), round-trip min/avg/max = 0/4/20 (ms)
```

Use Case 2: Check for the latest FTOS image and download and run script file for automated pre-requisite checks and, if pre-requisites are met, apply configuration

In this second use case, instead of pulling a configuration file from the DHCP server, we make things a little more dynamic by pulling an Expect script from a server and having it check some pre-configuration requirements. If the prerequisites are met, the configuration is applied, otherwise, it is not.

BMP supports Tcl, Expect, and Zsh scripts; in this use case we use Expect which is an extension to the Tcl scripting language. Expect is an extremely easy to learn and rapid development scripting language specifically designed for testing and automation in regards to interactive programs. It is preferable to use for the quickest results with the most minimal lines of code and time investment.

For this use case, the first thing we have to do is update the DHCP server 'dhcpd.conf' file to pull an Expect script to apply instead of just a configuration file. Code Snippet 2 below shows this update. You can see the only update made was to replace the referenced 'S4810-1.conf' config file with a reference to an Expect script, 'S4810-1-conf.exp'.

Code Snippet 2: 'dhcp.conf' from CentOS Linux server

```
#dhcpd.conf file

ddns-update-style none;
ignore client-updates;

option configfile code 209 = text;

default-lease-time 100000;   # 24 hours
max-lease-time 200000;       # 48 hours


#range of IP addresses for DHCP server use and additional options
subnet 10.11.128.0 netmask 255.255.252.0
{
   range 10.11.129.1 10.11.129.253;
   option domain-name-servers 10.11.127.1;
}


#the host to use the below values for will be matched via interface mac-address
host S4810-1
{
   #interface mac-address of switch that matches this host
   hardware ethernet 00:01:e8:8a:de:50;

   #ip address that DHCP server will assign host that has above interface mac-address
   fixed-address 10.11.129.215;

   #FTOS image path to check for latest FTOS image
   filename "ftp://labadmin:password@10.11.129.100/FTOS_Image.bin";

   #configuration file or script file to be applied to the host
   option configfile "ftp://labadmin:password@10.11.129.100/scripts/S4810-1-conf.exp ";
}
```

Now the more interesting part is definitely the Expect script, 'S4810-1-conf.exp', shown in its entirety in Code Snippet 3. As you can see the script is composed of multiple functions/procedures which each complete a specific task. All the functions except the first, which we'll discuss in a moment, are grouped in one of three groups:

1. Logging Functions
2. Prereq Check Functions
3. Config Functions

The 'Logging Functions' include functions that are used for logging the status/progress of the script/configuration to a file and sending it to a FTP server upon completion. The 'Prereq Check Functions' are used to check prerequisites the switch should meet before the full configuration is applied. The 'Config Functions' are used to complete the full configuration of the switch if the prerequisite checks pass. Note that there is no separate configuration file; the script creates the configuration on the fly if the prerequisite checks pass.

Note the '#!/usr/bin/expect' and '#/DELL-FORCE10' lines on the top of the script file are required. The first line is specific to where the script can find the Expect interpreter that will run the script. The second line is required only if the script file is a pre-configuration script file which it is in this case.

The very first function titled 'execFtosCommand' is a very important function called throughout the program. The function is passed an argument which contains a FTOS command to be run on the switch. The function runs the command and displays the resulting FTOS output. In order to accomplish this task a lot of details are taken care of behind the scenes by use of a special FTOS command used within the function.

Within the 'execFtosCommand' function, the Tcl 'exec' command is used to call a program from within the shell. FTOS is based on NetBSD so any supported Unix function/executable can be called via the Tcl 'exec' command. The program executed by 'exec' is actually a program called 'f10do'; this is a special program that takes an FTOS command as an argument and takes care of all the underlying details for us.

Once we get past all the function definitions, the program starts calling all the functions one by one in sequence until the end of the script. The full configuration gets applied under the section commented '#######Applying Config##########'. Note, if the prerequisite check fails, this section is never reached and the full configuration is not applied. The engineer can then check the log file sent to the FTP server to see what went wrong or if the script encountered an error.

The below screen capture displays the outputs/actions produced by the script if the prerequisite checks fail. We intentionally disconnected port 'TenGigabitEthernet 0/13' so the prerequisite check fails. As expected, the check fails, and the information is displayed on the terminal and written to a log file; the log file is then uploaded to a FTP server.

Figure 6: Script communicates prerequisite check failed and sends the log file to an FTP server

```
 Checking Conectivity via LLDP

show lldp neighbors | grep "Te 0/13 " ignore-case
FTOS#
     Interface is Not Connected

Wait for 20 seconds for the neighbor to come-up

show lldp neighbors | grep "Te 0/13 " ignore-case
FTOS#
     Interface is Not Connected

Wait for 20 seconds for the neighbor to come-up

show lldp neighbors | grep "Te 0/13 " ignore-case
FTOS#
     Interface is Not Connected

Wait for 20 seconds for the neighbor to come-up

ERROR: Interface TenGigabitEthernet 0/13 is NOT Connected

Error - LLDP Neighbor check failed!

Prerequisite check complete: requirements fail; configuration will not be applied!

Uploading Status File, (S4810-1-status.log), to 10.11.129.100 ...
```

Below is an example of the script communicating that the prerequisite checks have passed and that the configuration is now being applied. As you can see the configuration is only applied if the prerequisite checks pass.

Figure 7: Prerequiste check passes and script starts to configure switch

```
Checking if connected to the right switch!

Interface TenGigabitEthernet 0/1 is Connected to correct switch, S4810-3

Interface TenGigabitEthernet 0/1 is Connected

!!!!!Prerequisite check complete: requirements pass; configuration will now be applied!!!!!

configure terminal
FTOS(conf)#
          hostname S4810-1
S4810-1(conf)#
```

Below is the content of the entire script file used in this use case.

Code Snippet 3: 'S4810-1-conf.exp' Expect script

```
#!/usr/bin/expect

#/DELL-FORCE10

#Dell Networking
#BMP 3.0 - Automating the Network Whitepaper Script
#Use Case 2 Script - S4810-1-conf.exp
#Ver. 1.0
#02-08-2013


############FUNCTIONS############

#access the switch CLI via Dell 'f10do' shell command and execute FTOS command
proc execFtosCommand {cmd_str} {
   set str [exec f10do "$cmd_str"]
   print_output $str
}


########Logging Functions########

#print to terminal and to log file
proc print_output {str {fd ""}} {
   puts $str
   if {$fd != ""} {
      puts $fd $str
   }
}


#starts logging status
proc startLogging {} {
   global status_file
   global fp

   set status_file "S4810-1-status.log"

   #open status file
   set fp [open $status_file w]

}


#stop logging status
proc stopLogging {} {
   global fp

   print_output "=========================================\n" $fp
   close $fp
}


#FTP status log file to server
proc ftpLog {} {
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
global status_file

#ip address of FTP server and login credentials
set FTP_IP "10.11.129.100"
set FTP_USERNAME "labadmin"
set FTP_PASSWORD "password"

set FTP_LOGIN "ftp> "
set TIMEOUT 10
exp_log_user 0

print_output "Uploading Status File, ($status_file), to $FTP_IP ...\n"

if {[catch {exp_spawn ftp $FTP_IP} ret]} {
   set err true
   print_output "Failed to spawn ftp to $FTP_IP:\n\t$ret"
} else {
   set err false
}

if {!$err} {
   #Expect loop
   expect {
      timeout {
         print_output "Timed out waiting for login on $FTP_IP"
          }
          eof {
             print_output "FTP connection closed by $FTP_IP"
      }
      "Login failed" {
         print_output "FTP connection failed on $FTP_IP"
      }
      -re "ser (.*) denied" {
         print_output "FTP connection refused by $FTP_IP"
      }
          "Connection timed out" {
             print_output "FTP connection timed out on $FTP_IP"
      }
      "Unknown host " {
         print_output "Invalid host $FTP_IP"
      }
      -re "\n(User (.*): |Name (.*): )" {
             exp_send "$FTP_USERNAME\r"
             exp_continue
      }
      -re "\nPassword: *" {
         exp_send "$FTP_PASSWORD\r"
             exp_continue
      }
      -re "\n230 (.*) logged in" {
             exp_continue
      }
      -re $FTP_LOGIN {
      }
   }
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
        #do something with FTP
        exp_send "cd logs\r"
        exp_send "put $status_file\r"
        expect {
           timeout {
              print_output "Time out waiting for response on $FTP_IP"
           }
           eof {
              print_output "FTP connection closed by $FTP_IP"
           }
           -re $FTP_LOGIN {
                 print_output "Log file uploaded: /logs/$status_file "
           }
        }

        #end clean
        exp_send "quit\r"
        after 5000
        catch {exp_close}
    }
}


#####Prereq Check Functions######

#resets watchdog timer
proc resetTimer {} {
    print_output [exec rstimer 30]
    print_output "\nReset Timer Complete!\n"
}


#configures LLDP protocol
proc configLLDP {} {
    global fp

    print_output "Configuring LLDP Protocol!\n" $fp
    execFtosCommand "configure terminal"
    execFtosCommand "protocol lldp"
    execFtosCommand "no disable"
    execFtosCommand "advertise management-tlv system-name"
    execFtosCommand "end"
}


#confirms LLDP protocol is configured
proc confirmLLDP {} {
    global fp

    set lldp_output [exec f10do "show runn | grep lldp"]

    if {[regexp "protocol lldp" $lldp_output]} {
        print_output "LLDP successfully configured!\n" $fp
    } else {
        print_output "ERROR: LLDP configuration failed!\n" $fp
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
      #throw error
       error "LLDP configuration failed!!"
   }
}


#check LLDP neighbors
proc checkLLDP {passedList} {
   global fp

   upvar 1 [lindex $passedList 0] tempArray
   upvar 1 [lindex $passedList 1] tempArray2

   print_output "=========================================\n" $fp
   print_output " Checking Conectivity via LLDP\n" $fp
   print_output "=========================================\n" $fp

   #check via LLDP if ports are active and connected to right switch
   foreach port [array names tempArray] {
      set attempts 0
      set portPass 0
      set switchPass 0

      while {$portPass == 0 && $attempts < 3} {
         set splitArray [split $port]
         set portAbr [string range [lindex $splitArray 0] 0 1]
         set portAbr2 [lindex $splitArray 1]
         set finalAbrPort "${portAbr} ${portAbr2}"
         set result_str [exec f10do "show lldp neighbors | grep \"$finalAbrPort \" ignore-case"]

         print_output $result_str
         if {[regexp "$port" $result_str]} {
            set portPass 1
            if {[regexp "$tempArray2($port)" $result_str]} {
               print_output "Interface $port is Connected to $tempArray2($port)\n" $fp

               print_output "Checking if connected to the right switch!\n" $fp

               if {[regexp "$tempArray($port)" $result_str]} {
                  set switchPass 1
                  print_output "Interface $port is Connected to correct switch, $tempArray($port)\n" $fp
               } else {
                  print_output "ERROR: Interface $port is NOT Connected to correct switch,
$tempArray($port)\n" $fp
               }
            } else {
               print_output "ERROR: Interface $port is Not Connected to Interface $tempArray2($port)\n" $fp
               print_output "LLDP Output for $port:\n $result_str \n" $fp
            }
            continue
         }
         #wait for 20 seconds
         print_output "Interface is Not Connected\n"
         print_output "Wait for 20 seconds for the neighbor to come-up\n"
         after 20000
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
            incr attempts
        }

        if {$portPass == 1} {
            print_output "Interface $port is Connected\n" $fp
        } else {
            print_output "ERROR: Interface $port is NOT Connected\n" $fp

            #throw error
            error "LLDP Neighbor check failed!"
        }

        if {$switchPass != 1} {
            print_output "ERROR: Interface $port is NOT Connected to correct switch, $tempArray($port)\n" $fp

            #throw error
            error "LLDP Neighbor check failed!"
        }
    }
}


#configure interfaces used for prerequiste check
proc configInterfaces {passedArray} {
    global fp
    upvar 1 $passedArray tempArray

    foreach interface_value [array names tempArray] {
        print_output "Configuring $interface_value ...\n" $fp
        execFtosCommand "configure terminal"
        execFtosCommand "interface $interface_value"
        execFtosCommand "no ip address"
        execFtosCommand "no shutdown"
        execFtosCommand "end"
    }

        print_output "Wait for 10 seconds for the neighbors to come-up\n"
        after 10000
}


########Config Functions#########

#write changes to startup-config
proc doWrite {} {
    execFtosCommand "write"

    #wait 2 seconds
    after 2000
}


#set the hostname of the switch
proc setHostname {hostname} {
    execFtosCommand "configure terminal"
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
    execFtosCommand "hostname $hostname"
    execFtosCommand "end"
}


#sets IP addresses on respective interfaces
proc setIPs {passedArray} {
    global fp
    upvar 1 $passedArray tempArray

    foreach interface_value [array names tempArray] {
        print_output "Configuring $interface_value ...\n" $fp
        execFtosCommand "configure terminal"
        execFtosCommand "interface $interface_value"

        switch $interface_value {
            "TenGigabitEthernet 0/1" {
                set ip "10.1.0.1/30"
            }
            "TenGigabitEthernet 0/13" {
                set ip "10.0.0.1/30"
            }
            "fortyGigE 0/60" {
                set ip "103.0.0.2/30"
            }
        }

        execFtosCommand "ip address $ip"
        execFtosCommand "end"

    }
        print_output "Wait for 8 seconds for the Neighbors to come-up\n"
        after 8000
}


#configures loopback interface - only one loopback interface used for BGP
proc setLoopbacks {} {
    global fp

    print_output "Configuring Loopback 1 ...\n" $fp
    execFtosCommand "configure terminal"
    execFtosCommand "interface loopback 1"
    execFtosCommand "ip address 102.0.0.1/24"
    execFtosCommand "end"
}


#configures management interface with IP address
proc setManagement {} {
    global fp

    print_output "Configuring ManagementEthernet 0/0 ...\n" $fp
    execFtosCommand "configure terminal"
    execFtosCommand "interface managementEthernet 0/0"
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
    execFtosCommand "ip address 10.11.129.215/24"
    execFtosCommand "no shut"
    execFtosCommand "end"
}


#configures OSPF
proc configOSPF {} {
    global fp

    print_output "Configuring OSPF ...\n" $fp
    execFtosCommand "configure terminal"
    execFtosCommand "router ospf 5"
    execFtosCommand "router-id 255.0.0.0"
    execFtosCommand "y"
    execFtosCommand "network 10.0.0.0/30 area 0"
    execFtosCommand "network 10.1.0.0/30 area 0"
    execFtosCommand "end"
}


#configures BGP
proc configBGP {} {
    global fp

    print_output "Configuring BGP ...\n" $fp
    execFtosCommand "configure terminal"
    execFtosCommand "router bgp 5"
    execFtosCommand "network 102.0.0.0/16"
    execFtosCommand "neighbor 104.0.0.1 remote-as 10"
    execFtosCommand "neighbor 104.0.0.1 ebgp-multihop 255"
    execFtosCommand "neighbor 104.0.0.1 update-source Loopback 1"
    execFtosCommand "neighbor 104.0.0.1 no shutdown"
    execFtosCommand "end"
}


#configures routes - only one route configured for BGP
proc configRoutes {} {
    global fp

    print_output "Configuring routes ...\n" $fp
    execFtosCommand "configure terminal"
    execFtosCommand "ip route 104.0.0.0/16 103.0.0.1"
    execFtosCommand "end"
}


###################################

#translate CR and LF correctly
fconfigure stdout -translation crlf
fconfigure stderr -translation crlf

#flag variables to confirm if initial check and configuration pass/succeed
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
set checkFail 0
set configFail 0

#name to set the switch to
set hostname "S4810-1"

#values for respective interfaces used by script
set int1 "TenGigabitEthernet 0/1"
set int2 "TenGigabitEthernet 0/13"
set int3 "fortyGigE 0/60"

set interfaces($int1) "S4810-3"
set interfaces($int2) "S4810-2"
set interfaces($int3) "Z9000-1"

set remote_interfaces($int1) "TenGigabitEthernet 0/1"
set remote_interfaces($int2) "TenGigabitEthernet 0/13"
set remote_interfaces($int3) "fortyGigE 0/60"


print_output "!!!Executing Config Script!!!\n"


#set Unix watchdog timer to 30 minutes
if [catch resetTimer] {
    print_output "Error encountered during watchdog timer reset!\n"
    set checkFail 1
    exit 2
}


#start logging
if [catch startLogging] {
    print_output "Error encountered during start of logging!\n"
} else {
    print_output "Status logging successfully started!\n" $fp
}


print_output "=========================================\n" $fp
print_output "    Status: $hostname\n" $fp
print_output "=========================================\n" $fp


#configure LLDP protocol
if [catch configLLDP] {
    print_output "Error encountered during LLDP configuration!\n"
    set checkFail 1
}


#confirm LLDP protocol is configured
if [catch confirmLLDP] {
    print_output "Error - LLDP not configured!\n"
    set checkFail 1
}
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
#configure interfaces
if [catch {configInterfaces interfaces}] {
    print_output "Error upon inital interface configuration!\n"
    set checkFail 1
}


#check LLDP neighbors
if [catch {checkLLDP {interfaces remote_interfaces}}] {
    print_output "Error - LLDP Neighbor check failed!\n"
    set checkFail 1
}


#IF PREREQUISITE CHECK FAILS, EXIT SCRIPT, OTHERWISE CONTINUE TO APPLY CONFIGURATION
if {$checkFail} {
    print_output "!!!!!Prerequisite check complete: requirements fail; configuration will not be applied!!!!!\n" $fp

    #stop logging to status file
    if [catch stopLogging] {
        print_output "Error upon stopping logging!\n" $fp
    }

    #FTP status log file to server
    if [catch ftpLog] {
        print_output "\nError upon FTP of log file to server!\n"
    }

    exit 2
} else {
    print_output "!!!!!Prerequisite check complete: requirements pass; configuration will now be applied!!!!!\n"
$fp
}


#######Applying Config###########

#set the hostname of the switch
if [catch {setHostname $hostname}] {
    print_output "Error setting hostname!\n" $fp
    set configFail 1
}


#configure all physical interfaces that will be used with IP addresses
if [catch {setIPs interfaces}] {
    print_output "Error setting IPs on interfaces!\n" $fp
    set configFail 1
}


#configure loopback interfaces - only one loopback interface configured for BGP use
if [catch setLoopbacks] {
    print_output "Error configuring loopback addresses!\n" $fp
    set configFail 1
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
}


#configure OSPF
if [catch configOSPF] {
    print_output "Error configuring OSPF!\n" $fp
    set configFail 1
}


#configure BGP
if [catch configBGP] {
    print_output "Error configuring BGP!\n" $fp
    set configFail 1
}


#configure routes - only one route configured for BGP
if [catch configRoutes] {
    print_output "Error configuring routes!\n" $fp
    set configFail 1
}


#set the management IP address
if [catch setManagement] {
    print_output "Error configuring management!\n" $fp
    set configFail 1
}


#write the configuration to startup-config
if [catch doWrite] {
    print_output "Error saving configuration!\n" $fp
    set configFail 1
}


#NOTIFY IF CONFIGURATION FAILED OR PASSED
if {$configFail} {
    print_output "\n!!!!!Errors were encountered during configuration - check log file!!!!!\n" $fp
} else {
    print_output "\n!!!!!Configuration Successfully Applied!!!!!\n" $fp
}


#stop logging to status file
if [catch stopLogging] {
    print_output "Error upon stopping logging!\n" $fp
    exit 2
}


#FTP status log file to server
if [catch ftpLog] {
```

Code Snippet 3: 'S4810-1-conf.exp' Expect script cont.

```
    print_output "\nError upon FTP of log file to server!\n"
    exit 2
}


#IF CONFIGURATION FAILED, EXIT ON ERROR, OR, IF PASSED, EXIT ON SUCCESS
if {$configFail} {
    exit 2
} else {
    exit 0
}
```

Use Case 3: Check for the latest FTOS image and download and run script file for automated pre-requisite checks and, if pre-requisites are met, download and apply the full configuration file. Lastly, download and run a post-configuration script file to confirm successful configuration and status.

In this use case we have three files involved:

1. S4810-1-pre-conf.exp – prerequisite checks are performed
2. S4810-1.conf – configuration file that is downloaded and applied upon passing prerequisite checks
3. S4810-1-post-conf.exp – post-configuration checks are performed to make sure configuration is applied successfully and network status is ok

To support this new use case, we update the 'dhcpd.conf' file and replace the value for the 'option configfile' field to point to the new pre-configuration file, 'S4810-1-pre-conf.exp', shown further below in its entirety.

Example:
option configfile "ftp://labadmin:password@10.11.129.100/scripts/S4810-1-pre-conf.exp";

Below, we step through the details of each of the three files.

1. S4810-1-pre-conf.exp (See Code Snippet 4 further below)

In the pre-configuration script file, the respective interfaces are enabled with minimal layer 2 configuration, LLDP is configured, and the script confirms the correct connections are made by checking for LLDP neighbors. Success/failure results are captured in a log file and sent to a FTP server.

Notice in the script shown in its entirety further below in Code Snippet 4, the function, checkLLDP, checks not only the interface on the neighbor switch it is connected to but also the host name of the switch it is connected to. This ensures the switch is connected to not only the right port but also the right switch.

If the prerequisite checks fail, the configuration file is never downloaded and applied. The status log file is uploaded to the FTP server upon failure or success.

2. S4810-1.conf (See Code Snippet 5 further below)

Once the pre-configuration script finishes and the prerequisite checks pass, the full configuration file is downloaded. In Code Snippet 5 further below, only the relevant/applied configuration is shown. The configuration file has a command in it to run the post-configuration script file which was downloaded in the pre-configuration script:

script post-config /f10/flash/S4810-1-post-conf.exp

Once the configuration file is applied, the post-configuration script runs and confirms the configuration and status of the network is ok.

3. S4810-1-post-conf.exp (See Code Snippet 6 further below)

In the post-configuration script file, BGP neighbors, OSPF neighbors, and communication through the downstream VLT link is validated to confirm configuration was successful and no issue exists. Success/failure results are captured in a log file and sent to a FTP server.

Notice in the script shown in its entirety further below in Code Snippet 6, not only are BGP and OSPF neighbors checked by the 'checkBGP' and 'checkOSPF' functions respectively, but it is also verified that the neighbor is in the correct state. For BGP, it is also confirmed that the correct number of prefixes are learned.

Lastly, communication through the downstream VLT is checked via the 'ping' command in the 'checkVLT' function. The status log file is uploaded to the FTP server upon failure or success.

It is important to note that the pre-configuration script runs before the startup-config is loaded. The 'doWrite'' function in the pre-configuration script writes the changes made by the script to the startup-config. In this use case, if the prerequisites checked in the script pass, a complete configuration file is downloaded and copied to the startup-config. Within this downloaded configuration file is the configuration line that starts the execution of the post-configuration file once the startup-config is loaded: 'script post-config /f10/flash/S4810-1-post-conf.exp''.

The complete pre-configuration script is displayed below.

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
#!/usr/bin/expect

#/DELL-FORCE10

#Dell Networking
#BMP 3.0 - Automating the Network Whitepaper Script
#Use Case 3 Script - S4810-1-pre-conf.exp
#Ver. 1.0
#02-08-2013


############FUNCTIONS############

#access the switch CLI via Dell 'f10do' shell command and execute FTOS command
proc execFtosCommand {cmd_str} {
   set str [exec f10do "$cmd_str"]
   print_output $str
```

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
}


########Logging Functions########

#print to terminal and to log file
proc print_output {str {fd ""}} {
    puts $str
    if {$fd != ""} {
        puts $fd $str
    }
}


#starts logging status
proc startLogging {} {
    global status_file
    global fp

    set status_file "S4810-1-pre-status.log"

    #open status file
    set fp [open $status_file w]
}


#stop logging status
proc stopLogging {} {
    global fp

    print_output "=========================================\n" $fp
    close $fp
}


#FTP status log file to server
proc ftpLog {} {
    global status_file

    #ip address of FTP server and login credentials
    set FTP_IP "10.11.129.100"
    set FTP_USERNAME "labadmin"
    set FTP_PASSWORD "password"
    set FTP_LOGIN "ftp> "
    set TIMEOUT 10
    exp_log_user 0

    print_output "Uploading Status File, ($status_file), to $FTP_IP...\n"

    if {[catch {exp_spawn ftp $FTP_IP} ret]} {
        set err true
        print_output "Failed to spawn ftp to $FTP_IP:\n\t$ret"
    } else {
        set err false
    }
    if {!$err} {
```

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
#Expect loop
expect {
   timeout {
      print_output "Timed out waiting for login on $FTP_IP"
   }
   eof {
      print_output "FTP connection closed by $FTP_IP"
   }
   "Login failed" {
      print_output "FTP connection failed on $FTP_IP"
   }
   -re "ser (.*) denied" {
      print_output "FTP connection refused by $FTP_IP"
   }
   "Connection timed out" {
      print_output "FTP connection timed out on $FTP_IP"
   }
   "Unknown host " {
      print_output "Invalid host $FTP_IP"
   }
   -re "\n(User (.*): |Name (.*): )" {
      exp_send "$FTP_USERNAME\r"
      exp_continue
   }
   -re "\nPassword: *" {
      exp_send "$FTP_PASSWORD\r"
      exp_continue
   }
   -re "\n230 (.*) logged in" {
      exp_continue
   }
   -re $FTP_LOGIN {
   }
}

#do something with FTP
exp_send "cd logs\r"
exp_send "put $status_file\r"
expect {
   timeout {
      print_output "Time out waiting for response on $FTP_IP"
   }
   eof {
      print_output "FTP connection closed by $FTP_IP"
   }
   -re $FTP_LOGIN {
      print_output "Log file uploaded: /logs/$status_file"
   }
}

#end clean
exp_send "quit\r"
after 5000
catch {exp_close}
```

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
    }
}


#####Prereq Check Functions######

#resets watchdog timer
proc resetTimer {} {
    print_output [exec rstimer 30]
    print_output "\nReset Timer Complete!\n"
}


#configures LLDP protocol
proc configLLDP {} {
    global fp

    print_output "Configuring LLDP Protocol!\n" $fp
    execFtosCommand "configure terminal"
    execFtosCommand "protocol lldp"
    execFtosCommand "no disable"
    execFtosCommand "advertise management-tlv system-name"
    execFtosCommand "end"
}


#confirms LLDP protocol is configured
proc confirmLLDP {} {
    global fp

    set lldp_output [exec f10do "show runn | grep lldp"]

    if {[regexp "protocol lldp" $lldp_output]} {
        print_output "LLDP successfully configured!\n" $fp
    } else {
        print_output "ERROR: LLDP configuration failed!\n" $fp

        #throw error
        error "LLDP configuration failed!!"
    }
}


#check LLDP neighbors
proc checkLLDP {passedList} {
    global fp

    upvar 1 [lindex $passedList 0] tempArray
    upvar 1 [lindex $passedList 1] tempArray2

    print_output "=======================================\n" $fp
    print_output " Checking Conectivity via LLDP\n" $fp
    print_output "=======================================\n" $fp

    #check via LLDP if ports are active and connected to right switch
```

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
    foreach port [array names tempArray] {
        set attempts 0
        set portPass 0
        set switchPass 0

        while {$portPass == 0 && $attempts < 3} {
            set splitArray [split $port]
            set portAbr [string range [lindex $splitArray 0] 0 1]
            set portAbr2 [lindex $splitArray 1]
            set finalAbrPort "${portAbr} ${portAbr2}"

            set result_str [exec f10do "show lldp neighbors | grep \"$finalAbrPort \" ignore-case"]
            print_output $result_str
            if {[regexp "$port" $result_str]} {
                set portPass 1
                if {[regexp "$tempArray2($port)" $result_str]} {
                    print_output "Interface $port is Connected to $tempArray2($port)\n" $fp

                    print_output "Checking if connected to the right switch!\n" $fp

                    if {[regexp "$tempArray($port)" $result_str]} {
                        set switchPass 1
                        print_output "Interface $port is Connected to correct switch, $tempArray($port)\n" $fp
                    } else {
                        print_output "ERROR: Interface $port is NOT Connected to correct switch,
$tempArray($port)\n" $fp
                    }
                } else {
                    print_output "ERROR: Interface $port is Not Connected to Interface $tempArray2($port)\n" $fp
                    print_output "LLDP Output for $port:\n $result_str \n" $fp
                }
                continue
            }
            #wait for 20 seconds
            print_output "Interface is Not Connected\n"
            print_output "Wait for 20 seconds for the neighbor to come-up\n"
            after 20000
            incr attempts
        }

        if {$portPass == 1} {
            print_output "Interface $port is Connected\n" $fp
        } else {
            print_output "ERROR: Interface $port is NOT Connected\n" $fp

            #throw error
            error "LLDP Neighbor check failed!"
        }

        if {$switchPass != 1} {
            print_output "ERROR: Interface $port is NOT Connected to correct switch, $tempArray($port)\n" $fp

            #throw error
            error "LLDP Neighbor check failed!"
        }
```

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
    }
}


#configure interfaces used for prerequiste check
proc configInterfaces {passedArray} {
    global fp
    upvar 1 $passedArray tempArray

    foreach interface_value [array names tempArray] {
        print_output "Configuring $interface_value...\n" $fp
        execFtosCommand "configure terminal"
        execFtosCommand "interface $interface_value"
        execFtosCommand "no ip address"
        execFtosCommand "no shutdown"
        execFtosCommand "end"
    }
        print_output "Wait for 10 seconds for the neighbors to come-up\n"
        after 10000
}


#download and apply config file and post-config script
proc getConfigAndScript {} {
    set confFile "S4810-1.conf"
    set postConfScript "S4810-1-post-conf.exp"

    exec rm -rf "/usr/pkg/home/bmpuser/$confFile"
    exec rm -rf "/usr/pkg/home/bmpuser/$postConfScript"

    print_output "Downloading Startup Config and Post-Config Script...\n"

    catch {
        exec curl "ftp://labadmin:password@10.11.129.100/scripts/$confFile" -o $confFile
    }
    after 5000

    catch {
        exec curl "ftp://labadmin:password@10.11.129.100/scripts/$postConfScript" -o $postConfScript
    }
    after 5000

    exec cp -f "$confFile" "/f10/flash/$confFile"
    after 5000

    exec cp -f "$postConfScript" "/f10/flash/$postConfScript"
    after 5000

    print_output "Download Complete!!!\n"


    if {[file exists $confFile]} {
        print_output "Config File: $confFile downloaded successfully\n"
    } else {
        print_output "ERROR: Config File: $confFile - Not Found\n"
    }
```

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
    if {[file exists $postConfScript]} {
        puts "Post Config Script: $postConfScript downloaded successfully\n"
    } else {
       puts "ERROR: Post Config Script: $postConfScript - Not Found\n"
    }
}


########Config Functions#########

#write changes to startup-config
proc doWrite {} {
    global fp

    execFtosCommand "copy S4810-1.conf startup-config"
    execFtosCommand "yes"
}

###################################

#translate CR and LF correctly
fconfigure stdout -translation crlf
fconfigure stderr -translation crlf

#flag variables to confirm if initial check and configuration pass/succeed
set checkFail 0
set configFail 0

#name to set the switch to
set hostname "S4810-1"

#values for respective interfaces used by script
set int1 "TenGigabitEthernet 0/1"
set int2 "TenGigabitEthernet 0/13"
set int3 "fortyGigE 0/60"

set interfaces($int1) "S4810-3"
set interfaces($int2) "S4810-2"
set interfaces($int3) "Z9000-1"

set remote_interfaces($int1) "TenGigabitEthernet 0/1"
set remote_interfaces($int2) "TenGigabitEthernet 0/13"
set remote_interfaces($int3) "fortyGigE 0/60"


puts "\n!!!!!!!!!!Executing Pre-Config Script and Checking Prerequisites!!!!!!!!!!\n"

#set Unix watchdog timer to 30 minutes
if [catch resetTimer] {
    puts "Error encountered during watchdog timer reset!"
    set checkFail 1
    exit 2
}


#start logging
```

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
if [catch startLogging] {
    print_output "Error encountered during start of logging!\n"
} else {
    print_output "Status logging successfully started!\n" $fp
}


print_output "=========================================\n" $fp
print_output "    Status: $hostname\n" $fp
print_output "=========================================\n" $fp


#configure LLDP protocol
if [catch configLLDP] {
    print_output "Error encountered during LLDP configuration!\n"
    set checkFail 1
}


#confirm LLDP protocol is configured
if [catch confirmLLDP] {
    print_output "Error - LLDP not configured!\n"
    set checkFail 1
}


#configure interfaces
if [catch {configInterfaces interfaces}] {
    print_output "Error upon inital interface configuration!\n"
    set checkFail 1
}


#check LLDP neighbors
if [catch {checkLLDP {interfaces remote_interfaces}}] {
    print_output "Error - LLDP Neighbor check failed!\n"
    set checkFail 1
}


#IF PREREQUISITE CHECK FAILS, EXIT SCRIPT, OTHERWISE CONTINUE TO APPLY CONFIGURATION
if {$checkFail} {
    print_output "!!!!!Prerequisite check complete: requirements fail; configuration will not be applied!!!!!\n" $fp

    #stop logging to status file
    if [catch stopLogging] {
        print_output "Error upon stopping logging!\n" $fp
    }

    #FTP status log file to server
    if [catch ftpLog] {
        print_output "\nError upon FTP of log file to server!\n"
    }

    exit 2
```

Code Snippet 4: 'S4810-1-pre-conf.exp' Expect script cont.

```
} else {
   print_output "!!!!!Prerequisite check complete: requirements pass; configuration will now be applied!!!!!\n"
$fp
}


#######Applying Config###########

if [catch getConfigAndScript] {
   print_output "Error downloading configuration and script file!" $fp
   set configFail 1
}


#copy the configuration to startup-config
if [catch doWrite] {
   print_output "Error saving downloaded configuration to startup-config!\n" $fp
   set configFail 1
}


#NOTIFY IF CONFIGURATION FAILED OR PASSED
if {$configFail} {
   print_output "\n!!!!!Errors were encountered during configuration - check log file!!!!!\n" $fp
} else {
   print_output "\n!!!!!Configuration Successfully Applied!!!!!\n" $fp
}


#stop logging to status file
if [catch stopLogging] {
   print_output "Error upon stopping logging!\n" $fp
   exit 2
}


#FTP status log file to server
if [catch ftpLog] {
   print_output "\nError upon FTP of log file to server!\n"
   exit 2
}


#IF CONFIGURATION FAILED, EXIT ON ERROR, OR, IF PASSED, EXIT ON SUCCESS
if {$configFail} {
   exit 2
} else {
   exit 0
}
```

Once the above pre-configuration script completes and all the prerequisite checks pass, the below configuration file that was downloaded via the pre-configuration script is applied. Note, in Code Snippet 5 shown below, only the relevant/applied configuration is shown. If prerequisite checks fail, the configuration is not applied and the status log file is sent to the FTP server for the network engineer to review.

The post-configuration script shown further below in Code Snippet 6 is also downloaded via the pre-configuration script. Once the configuration file is applied, the 'script post-config /f10/flash/S4810-1-post-conf.exp' command in the configuration file executes the post-configuration script. On completion, the post-configuration script uploads the results of its post-configuration checks to the FTP server.

Code Snippet 5: 'S4810-1-conf' Configuration.

```
hostname S4810-1
!
!.....
!
interface TenGigabitEthernet 0/1
 ip address 10.1.0.1/30
 flowcontrol rx on tx off
 no shutdown
!
!.....
!
interface TenGigabitEthernet 0/13
 ip address 10.0.0.1/30
 flowcontrol rx on tx off
 no shutdown
!
!.....
!
interface fortyGigE 0/60
 ip address 103.0.0.2/30
 flowcontrol rx on tx off
 no shutdown
!
interface ManagementEthernet 0/0
 ip address 10.11.129.215/22
 no shutdown
!
!.....
!
interface Loopback 1
 ip address 102.0.0.1/24
 no shutdown
!
!.....
!
script post-config /f10/flash/S4810-1-post-conf.exp
!
router ospf 5
 router-id 255.0.0.0
 network 10.0.0.0/30 area 0
```

Code Snippet 5: 'S4810-1-conf' Configuration.

```
 network 10.1.0.0/30 area 0
!
router bgp 5
 network 102.0.0.0/16
 neighbor 104.0.0.1 remote-as 10
 neighbor 104.0.0.1 ebgp-multihop 255
 neighbor 104.0.0.1 update-source Loopback 1
 neighbor 104.0.0.1 no shutdown
!
ip route 104.0.0.0/16 103.0.0.1
!
protocol lldp
 advertise management-tlv system-name
```

Code Snippet 6: 'S4810-1-post-conf.exp' Expect script cont.

```
#!/usr/bin/expect

#Dell Networking
#BMP 3.0 - Automating the Network Whitepaper Script
#Use Case 3 Script - S4810-1-post-conf.exp
#Ver. 1.0
#02-08-2013


############FUNCTIONS############

#access the switch CLI via Dell 'f10do' shell command and execute FTOS command
proc execFtosCommand {cmd_str} {
    set str [exec f10do "$cmd_str"]
    print_output $str
}


########Logging Functions########

#print to terminal and to log file
proc print_output {str {fd ""}} {
    puts $str
    if {$fd != ""} {
        puts $fd $str
    }
}


#starts logging status
proc startLogging {} {
    global status_file
    global fp

    set status_file "S4810-1-post-status.log"
    #open status file
    set fp [open $status_file w]
```

Code Snippet 6: 'S4810-1-post-conf.exp' Expect script cont.

```
}


#stop logging status
proc stopLogging {} {
   global fp
   print_output "=========================================\n" $fp
   close $fp
}


#FTP status log file to server
proc ftpLog {} {
   global status_file

   #ip address of FTP server and login credentials
   set FTP_IP "10.11.129.100"
   set FTP_USERNAME "labadmin"
   set FTP_PASSWORD "password"
   set FTP_LOGIN "ftp> "
   set TIMEOUT 10
   exp_log_user 0

   print_output "Uploading Status File, ($status_file), to $FTP_IP...\n"

   if {[catch {exp_spawn ftp $FTP_IP} ret]} {
      set err true
      print_output "Failed to spawn ftp to $FTP_IP:\n\t$ret"
   } else {
      set err false
   }

   if {!$err} {
      #Expect loop
      expect {
         timeout {
            print_output "Timed out waiting for login on $FTP_IP"
         }
         eof {
            print_output "FTP connection closed by $FTP_IP"
         }
         "Login failed" {
            print_output "FTP connection failed on $FTP_IP"
         }
         -re "ser (.*) denied" {
            print_output "FTP connection refused by $FTP_IP"
         }
         "Connection timed out" {
            print_output "FTP connection timed out on $FTP_IP"
         }
         "Unknown host " {
            print_output "Invalid host $FTP_IP"
         }
         -re "\n(User (.*): |Name (.*): )" {
            exp_send "$FTP_USERNAME\r"
```

Code Snippet 6: 'S4810-1-post-conf.exp' Expect script cont.

```
            exp_continue
         }
         -re "\nPassword: *" {
            exp_send "$FTP_PASSWORD\r"
            exp_continue
         }
         -re "\n230 (.*) logged in" {
            exp_continue
         }
         -re $FTP_LOGIN {
         }
      }

      #do something with FTP
      exp_send "cd logs\r"
      exp_send "put $status_file\r"
      expect {
         timeout {
            print_output "Time out waiting for response on $FTP_IP"
         }
         eof {
            print_output "FTP connection closed by $FTP_IP"
         }
         -re $FTP_LOGIN {
            print_output "Log file uploaded: /logs/$status_file"
         }
      }

      #end clean
      exp_send "quit\r"
      after 5000
      catch {exp_close}
   }
}


#####Post-Config Check Functions######

#resets watchdog timer
proc resetTimer {} {
   print_output [exec rstimer 30]
   print_output "\nReset Timer Complete!\n"
}


#check BGP neighbors
proc checkBGP {bgpArray} {
   global fp

   upvar 1 $bgpArray tempArray

   print_output "\n!!!Checking BGP Neighbors!!!\n" $fp

   #check if BGP neighbor(s) are established and if correct # of prefixes learned
   foreach neighbor [array names tempArray] {
```

Code Snippet 6: 'S4810-1-post-conf.exp' Expect script cont.

```
        set result_str [exec f10do "show ip bgp neighbors $neighbor"]
        print_output $result_str $fp

        if {[regexp "BGP state ESTABLISHED" $result_str]} {
            print_output "\nBGP neighbor $neighbor has 'established' state!\n" $fp
            print_output "\nBGP Prefixes Accepted: $tempArray($neighbor)\n" $fp

            if {[regexp "Prefixes accepted $tempArray($neighbor)" $result_str]} {
                print_output "BGP neighbor $neighbor has correct number of BGP prefixes accepted!\n" $fp
            } else {
                print_output "\nBGP neighbor $neighbor does NOT have correct number of BGP prefixes
accepted!\n" $fp
                error "BGP neighbor check failed!"
            }
        } else {
            print_output "\nBGP neighbor $neigbor does NOT have 'established' state!" $fp
            error "BGP neighbor check failed!"
        }
    }
}


#check OSPF neighbors
proc checkOSPF {ospfArray} {
    global fp
    upvar 1 $ospfArray tempArray

    print_output "\n!!!Checking OSPF Neighbors!!!\n" $fp

    #check if OSPF neighbor(s) are learned and in 'FULL' state
    foreach neighbor [array names tempArray] {
        set result_str [exec f10do "show ip ospf neighbor | grep \"$neighbor\""]
        print_output $result_str
        if {[regexp "$neighbor" $result_str]} {
            print_output "\nOSPF neighbor $neighbor learned!\n" $fp

            if {[regexp "FULL" $result_str]} {
                print_output "\nOSPF neighbor $neighbor state: FULL\n" $fp
            } else {
                print_output "\nOSPF neighbor $neighbor state: NOT FULL\n" $fp
                error "OSPF neighbor $neighbor is NOT in FULL state!"
            }
        } else {
                print_output "\nOSPF neighbor $neighbor NOT learned!\n" $fp
                error "OSPF neighbor $neighbor NOT learned!!"
        }
    }
}


#check communication through downstream VLT
proc checkVLT {vlt_address} {
    global fp

    upvar 1 $vlt_address vlt_ip
```

Code Snippet 6: 'S4810-1-post-conf.exp' Expect script cont.

```
    print_output "\n!!!Checking communication through downstream VLT!!!\n" $fp

    set result_str [exec f10do "ping $vlt_ip"]
    print_output "$result_str\n" $fp

    if {[regexp "Success rate is 100.0 percent" $result_str]} {
        print_output "Communication through downstream VLT: Success\n" $fp
    } else {
        print_output "Communication through downstream VLT: Failed\n" $fp
        error "Communication through downstream VLT failed"
    }
}


###################################

#translate CR and LF correctly
fconfigure stdout -translation crlf
fconfigure stderr -translation crlf

#flag variables to confirm if initial check and configuration pass/succeed
set checkFail 0

#name to set the switch to
set hostname "S4810-1"

#values for respective neighbors
set bgp_neighbors(104.0.0.1) "11"
set ospf_neighbors(2.0.0.0) "10.0.0.2"
set ospf_neighbors(3.0.0.0) "10.1.0.2"
set vltIP "10.3.0.3"


print_output "\n!!!!!!!!!!Executing Post-Config Script!!!!!!!!!!\n"


#set Unix watchdog timer to 30 minutes
if [catch resetTimer] {
    print_output "Error encountered during watchdog timer reset!"
    set checkFail 1
    exit 2
}


#start logging
if [catch startLogging] {
    print_output "Error encountered during start of logging!\n"
} else {
    print_output "Status logging successfully started!\n" $fp
}


print_output "\n=========================================\n" $fp
print_output "    Status: $hostname\n" $fp
print_output "=========================================\n" $fp
```

Code Snippet 6: 'S4810-1-post-conf.exp' Expect script cont.

```
#wait 30 seconds for convergence
print_output "Waiting 30 seconds for convergence" $fp
after 30000


#check BGP
if [catch {checkBGP bgp_neighbors}] {
    print_output "BGP neighbor check failed!\n" $fp
    set checkFail 1
} else {
    print_output "BGP neighbor check passed!\n" $fp
}

after 5000

#check OSPF
if [catch {checkOSPF ospf_neighbors}] {
    print_output "OSPF neighbor check failed!\n" $fp
    set checkFail 1
} else {
    print_output "OSPF neighbor check passed!\n" $fp
}

after 5000

#check communication through downstream VLT
if [catch {checkVLT vltIP}] {
    print_output "VLT communication check failed!\n" $fp
    set checkFail 1
} else {
    print_output "VLT communication check passed!\n" $fp
}


###Confirm if Post-Config Passed###

#NOTIFY IF POST-CONFIG CHECK FAILED OR PASSED
if {$checkFail} {
    print_output "\n!!!!!Post-Configuration Check Failed!!!!!\n" $fp
} else {
    print_output "\n!!!!!Post-Configuration Check Passed!!!!!\n" $fp
}


#stop logging to status file
if [catch stopLogging] {
    print_output "Error upon stopping logging!\n" $fp
    exit 2
}


#FTP status log file to server
if [catch ftpLog] {
    print_output "\nError upon FTP of log file to server!\n"
    exit 2
```

Code Snippet 6: 'S4810-1-post-conf.exp' Expect script cont.

```
}


#IF POST-CONFIG CHECK FAILED, EXIT ON ERROR, OR, IF PASSED, EXIT ON SUCCESS
if {$checkFail} {
   exit 2
} else {
   exit 0
}
```

Figure 8 below displays a partial snapshot of the post-configuration log file uploaded to the FTP server during post-configuration script execution.

Figure 8: Partial snapshot of post-configuration log file

```
Status logging successfully started!


========================================

    Status: S4810-1

========================================

Waiting 30 seconds for convergence

!!!Checking BGP Neighbors!!!

show ip bgp neighbors 104.0.0.1


BGP neighbor is 104.0.0.1, remote AS 10, external link
  BGP version 4, remote router ID 115.0.0.1
  BGP state ESTABLISHED, in this state for 00:00:53
  Last read 00:00:52, last write 00:00:52
  Hold time is 180, keepalive interval is 60 seconds
  Received 4 messages, 0 in queue
    1 opens, 0 notifications, 1 updates
    2 keepalives, 0 route refresh requests
  Sent 3 messages, 0 in queue
    1 opens, 0 notifications, 0 updates
    2 keepalives, 0 route refresh requests
  Minimum time between advertisement runs is 30 seconds
  Minimum time before advertisements start is 0 seconds

  Capabilities received from neighbor for IPv4 Unicast :
    MULTIPROTO_EXT(1)
    ROUTE_REFRESH(2)
    CISCO_ROUTE_REFRESH(128)

  Capabilities advertised to neighbor for IPv4 Unicast :
    MULTIPROTO_EXT(1)
    ROUTE_REFRESH(2)
    CISCO_ROUTE_REFRESH(128)

  EBGP-multihop enabled, multihop TTL set to 255
  Update source set to Loopback 1

  For address family: IPv4 Unicast
  BGP table version 11, neighbor version 11
  Prefixes accepted 11 (consume 44 bytes), withdrawn 0 by peer, martian prefixes ignored 0
  Prefixes advertised 0, denied 0, withdrawn 0 from peer
```

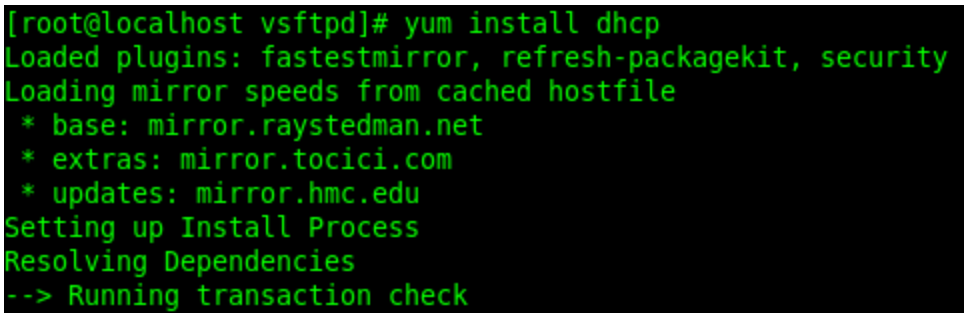
Conclusion

In this whitepaper, we demonstrated some powerful uses of BMP 3.0. The new feature to allow scripts to automate testing and configuration on switch boot-up allows for maximizing productivity and reliability by drastically minimizing user intervention/errors. Although we've demonstrated some popular use cases in this whitepaper such as configuring multiple protocols, testing for connectivity, and using FTP to send result data, the variation in the types of scripts that can be written and their respective behavior are numerous and left to the needs/requirements of the customer. As shown in the three use cases in this whitepaper, BMP 3.0 provides the flexibility to meet different requirements from downloading a simple configuration file to automating testing and conditional configuration based on the network status.

Appendix A: Installing a DHCP server

For a DHCP server, we used 'dhcpd', the Internet Systems Consortium DHCP Server which is available as open source software with an ISC License, a BSD style license. We used CentOS 6.3 for the operating system. Simply install the DHCP server via the 'yum install dhcp' command as shown below.

Figure 9: Installing 'dhcpd' on CentOS



Once the 'dhcpd' program is installed, we only need to edit the 'dhcpd.conf' file. The full path of the file location is '/etc/dhcp/dhcpd.conf'. To edit this file, you will need to be the 'root'user. To change to the 'root' user simply enter the 'su' command and enter the password you set for 'root' upon install. See the prior use case examples in this whitepaper to see how we edited the 'dhcpd.conf' file. Always make sure to create a backup copy of the 'dhcpd.conf' file before making changes.

Once the 'dhcpd.conf' file is updated, it will not take effect until the DHCP service is restarted. To accomplish this without having to reboot the system, use the below commands to stop, start, or restart the service. The DHCP service should be restarted every time after a change is made to the 'dhcp.d' file. You can accomplish this by either method below.

1.) service dhcpd restart

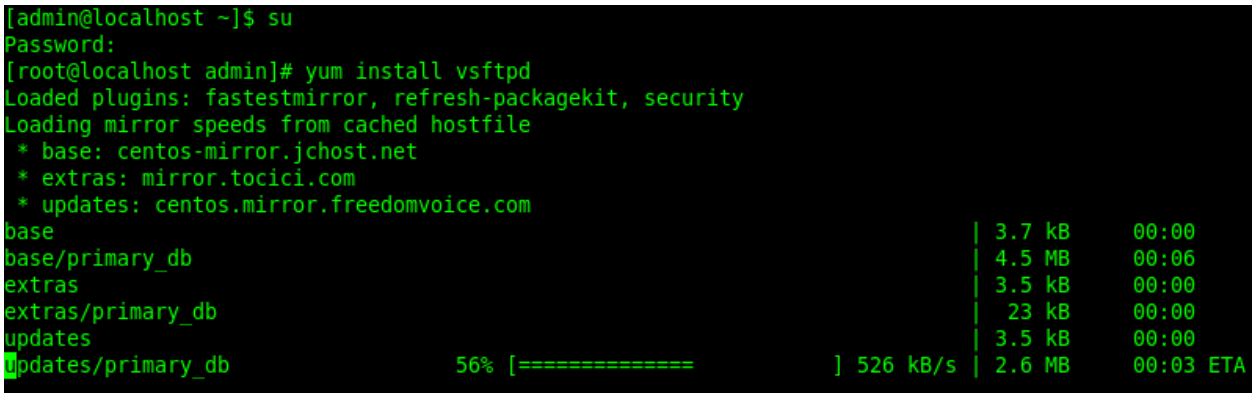
or

2.) service dhcpd stop
service dhcpd start

Appendix B: Installing a FTP server

For a FTP server, we used ‘vsftpd’, a GPL licensed FTP server for Unix/Linux systems. We used CentOS 6.3 for the operating system. Setting up a FTP server is fairly easy using the 'yum install vsftpd' command as 'root' via the command line interface. See the below snapshot. We switch the user to root using the 'su' command and then install 'vsftpd' using the 'yum install vsftpd' command.

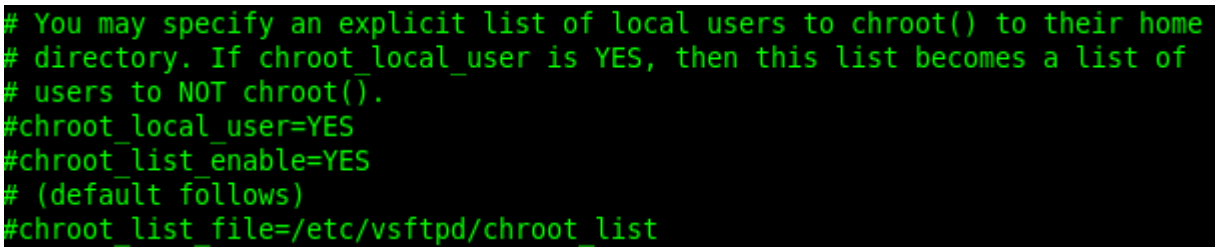
Figure 10: Installing 'vsftpd' on CentOS



Once the 'vsftpd' program is installed, we need to make a few changes.

As root, we edit the '/etc/vsftpd/vsftpd.conf' file to 'chroot' or 'jail' users to their home directories. This enforces security and privacy as well as enables us to simply upload files to our home directory, '/home/admin' to be used by BMP. Always make sure to create a backup copy of the ‘vsftpd.conf’ file before making changes. You will see something like the below in the file.

Figure 11: Editing the 'vsftpd.conf' file



Make sure to edit the file so all lines starting with ‘chroot’ are uncommented; this can be done by removing the ‘#’ symbol from the start of the respective line. Once the file is saved, set 'SELinux' to enforcing mode via the below command at the command prompt:

- setsebool -P ftp_home_dir=1

Now we can simply restart the FTP server via the ‘service dhcpd restart’ command. The ‘vsftpd’ service should be restarted every time after a change is made to the ‘vsftpd’ file. The below commands can be used for this purpose. You can accomplish this by either method below.

1.) service vsftpd restart

or

2.) service vsftpd stop
service vsftpd start