OpenFlow Assisted Virtual Network Migration on the VirtualBox 4.0 Hypervisor

Nikhil Khatu

CSC/ECE 570
Professor Thuente
12/11/2012

Project Description

Today, it is increasingly common to virtualize host machines running on well known hypervisors such VMware, VirtualBox, or Xen. Migrating Virtual Hosts is standard; we're now at the next evolutionary stage in networking in which networking equipment may also be virtualized. This virtualization and migration brings many complexities and unanswered questions with it.

These can be addressed with the advent of OpenFlow. The project takes the approach of migrating virtualized networking components with the added complexities. The added complexities can be addressed with the centralized OpenFlow Management used to preserve traffic flow during migration.

During the design phase previous implementations will be considered to create a prototype networking system. An additional feature that is addressed in this project design is migration over WAN (different subnetworks) which brings additional complexities.

Objectives

The objectives are as follows:

- Design a prototype network that is capable of being migrated
- Test Controller functionality with the prototype network(i.e. Mininet)
- Create the Host/Guest Environment on Oracle VirtualBox HyperVisor
- Implement Network migration with the VirtualBox 'teleport' feature
- Preserve network functionality in the migrated network
- Preserve traffic flows and data with OpenFlow

Summary of relevant papers

Previous research has been conducted to gauge performance during migration. In the project's context of migration it helps to extract research of network migration implementations. Although migration of networking components is akin to migrating virtual guests on a host system in terms of

virtual hardware performance; migrating networking components have a direct performance impact on the network as a whole.

When hosts are migrated across a network or to other networks the networking domains are relatively static which has this predictable factor. In “Live Migration of an Entire Software-Defined Network” [6] communication between migrated hosts use tunneling algorithms ASAP and ALAP. In both of these tunneling algorithms static switches are a presumption. Another implementation- Cloud Rack (based on the previous Mobitopolo concept) uses its Topology Central Control Service(TCCS) in a client-server model that runs alongside OpenvSwitch and OpenFlow Controller processes [5]. Although very useful, the concept is directed towards migrating/controlling just the traffic flow as opposed to network migration. In “OpenFlow Supporting Inter-Domain Virtual Machine Migration” the network topology is proposed as “abstract” and residing on the IaaS [4]. These VMs are organized based on hierarchy levels and then migrated based on “Global rules” and “Specific rules.” These rules are translated to create OpenFlow control. The network is presumed to be static here as well. IaaS topology virtualization applies to this migration project in the fact that the prototype is fully virtualized. In the research paper “Live Migration of Virtual Machines” a study is conducted of the impact on Virtual resources during live migrations [3].

While flow algorithms are useful for data transfer during migration they have little to say when implementing a network migration(differentiating from host migration). Two research papers bring light to Migrating a Software Defined Network; "Transparent, Live Migration of a Software-Defined Network" (Ghorbani, Soudeh, et al.) [2] and "Openflow and Xen-based virtual network migration” (Pisa, Pedro, et al.) [1].

In "Transparent, Live Migration of a Software-Defined Network” network migration is defined as “Ensemble Migration” where the (Live Migration of Ensembles) LIME prototype has been implemented. In order to provide “seemless” migration LIME uses two base topologies that provide redundancy. In the “Tree” topology the hosts are paired with the switch during migration. In the “Chain” topology the string of switches are all migrated together. Interestingly, LIME optimizes “freeze and copy” during teleportation of the SDN switch state by incrementing the process; thus reducing downtime. The increment optimization is gained due to state changes being table entries (differentiating from the traditional VM for hosts).In addition to “approach”(switch moving/cloning) algorithms LIME provides customized algorithms for “order” and “grouping.”

While LIME provides a customized way to migrate a network ensemble, these migrations are with respect to LAN toplogies. "Openflow and Xen-based virtual network migration” [1] fills a gap by introducing SDN routing in the core. Two key concepts tested in this research include: standard machine migration and data/control plane separation. Observations here show data/control planes seperation is key to reducing packet loss during migration, while Openflow provides a downtime of less than 5 milliseconds with no packet loss [1].

System design

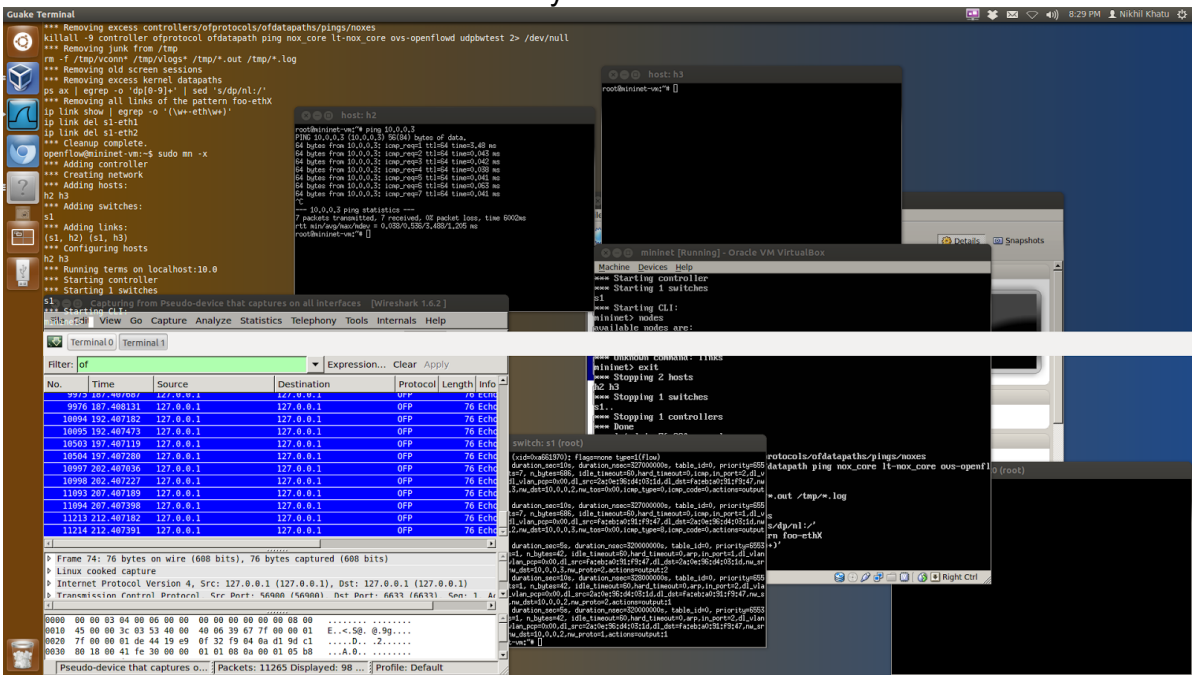
The design process follows the objective flow.

Design a prototype network that is capable of being migrated

Migrating algorithms were taken into account to create a base network. The base network includes two virtual hosts connected to a virtual SDN switch. The virtual SDN switch is connected to a virtualized “static” router. The base network of two virtual hosts and a virtual SDN switch should be migrated across the router to another subnetwork. The topology is depicted in the Scenarios section below.

Test Controller functionality with the prototype network(i.e. Mininet)

As a precursor to the project design implementation there are many tools available to simulate network functionality.



The screenshot above is of the Mininet walkthrough tutorial provided by Stanford's OpenFlow initiative [10]. An Ubuntu Virtual Machine is provided. Once the Virtual Disk Image is mounted to the preferred hypervisor and booted one can run network simulations for use with the OpenFlow controller of choice. (E.g. Pox, Nox, Floodlight, or Beacon)

Create the Host/Guest Environment on Oracle VirtualBox HyperVisor

The Hypervisor is configured according to the User Manual [7].
The project prototype system specifications:

- Intel i7 3.1 GHz core w/ built in GPU
- 16GB RAM -120 GB SSD Solid State Drive
- Host Environment: Ubuntu 12.04 LTS 64bit
- Oracle 4.0 Hypervisor with Guest Environment: Ubuntu 10.04 LTS 64bit
- OpenvSwitch 1.4.3 - OpenFlow Pox Controller

***** NOTE: Host environment must have available resources for the Guest Machines**

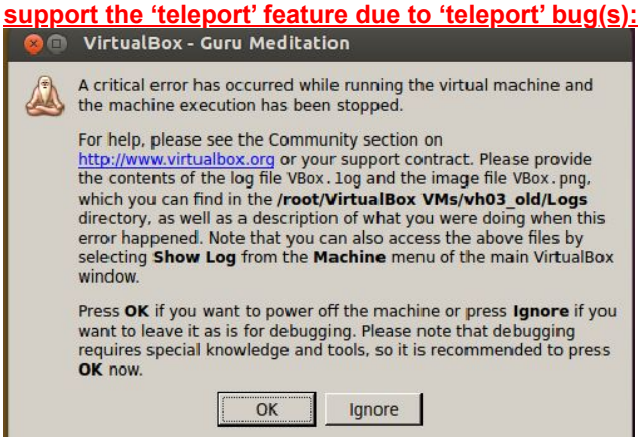
Bash script files are pasted in appendix:
create_original_vh01.sh
create_original_vh02.sh
create_original_vh03.sh
Once the VMs are created with the script files we can start them (sudo VBoxManage startvm "<vm_name>") and install the desired guest environment.

Implement Network migration with the VirtualBox 'teleport' feature

Migration was implemented with Trial and Error. Import notes are listed in red below.
Maintaining the bash script files was integral to success.

Run the scripts on remote VBox hypervisor(ports , vm name, etc can be adjusted):
****NOTE: Remote VBox hypervisor must share the same disk image.(e.g. remote iSCSI/NFS, localhost definition) Edit script files to specify listener port and vdi file)**
make_vh01_clone.sh
make_vh02_clone.sh
make_vh03_clone.sh

*****NOTE: Host environment must use VBox 4.0. Current VBox releases 4.1 and 4.2 do not support the 'teleport' feature due to 'teleport' bug(s):**



*****NOTE: VBox 4.0 doesn't have a check box for "promiscous mode" on interfaces. Must manually enable the variable:**

VBoxManage setextradata "<vm_name>" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/IfPolicyPromisc" "allow-all"

CAUTION! MUST ONLY ENABLE PROMISCUOUS MODE ON INTERNAL ENABLED INTERFACES!!! USE PCI III INTERFACE NIC ONLY!!! INTERFACE 1 = '/0/', INTERFACE 2 = '/1/', ETC...

Failure to do so will cause boot failure. Reverse command is unknown.

Once configuration is confirmed run Bash script migrate.sh to migrate VM. Edit VMname, IP, and destination port as needed.

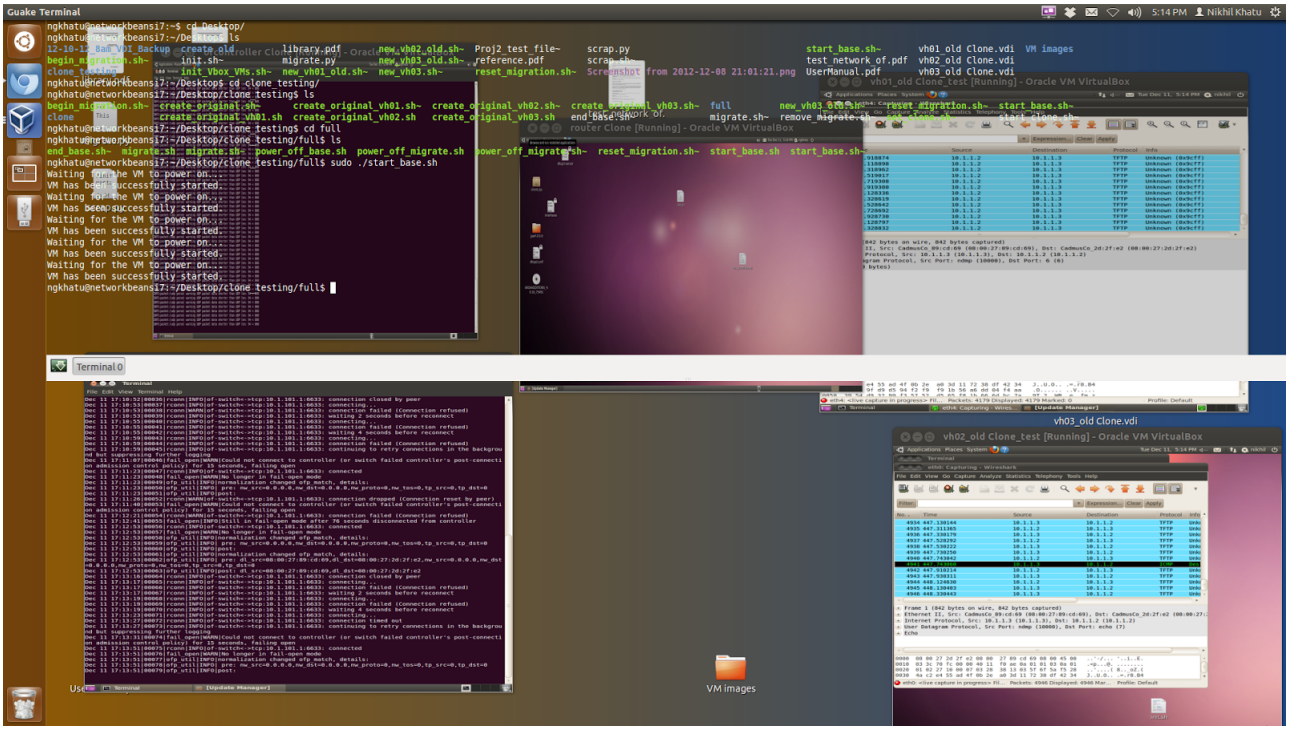
Preserve network functionality in the migrated network

DHCP and IP_Forwarding needs to be configured on the router and Controller machines.
DHCP allocates a fixed IP for each host on each subnet. The controller is attached to the "management" subnet.

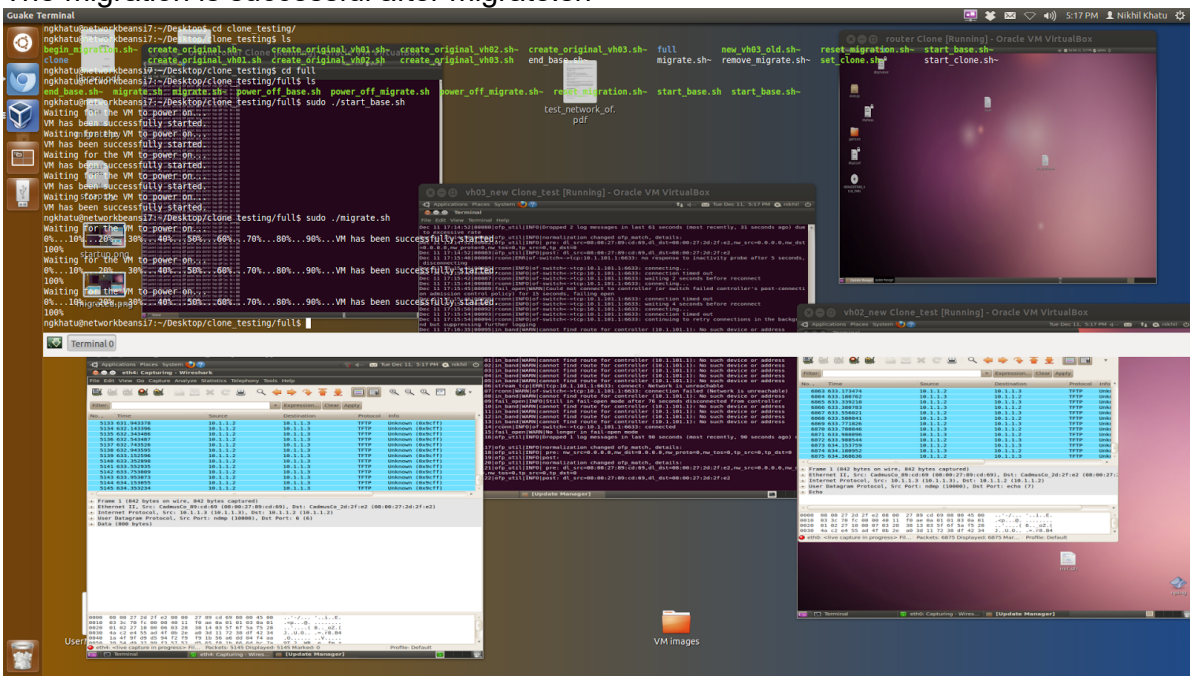
Traffic can be generated with a packet generator to test functionality. (e.g. packETH, nping, ping, iperf, jperf, etc...)

Use these bash scripts to debug the setup and demo:
start_base.sh
migrate.sh
power_off_base.sh
power_off_migrate.sh

The Base VMs have now been started with start_base.sh:



The migration is successful after migrate.sh



***NOTE: Currently the VH must send a DHCP request after migration, therefore the migration is not seamless.

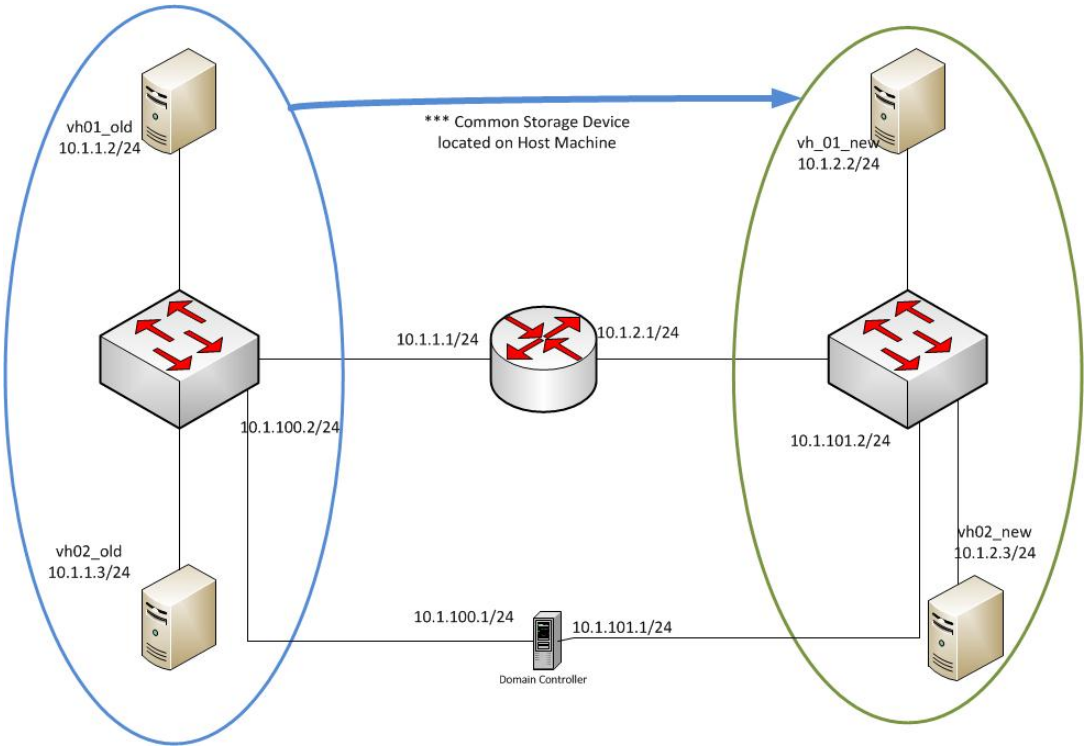
Preserve traffic flows and data with OpenFlow

Currently vh03_old must run in standalone mode to maintain normal l2_learning/switching.
Stanford's online OpenFlow Tutorial [8] is used to define a l2_learning switch and a demo flow is created using the Pox Wiki [9]. To demo (the demo algorithm is discussed in the section below) the modified flow created with Pox OpenFlow Controller:

- Start the VMs on host machine: *sudo ./start_base.sh*
- Run on the ofcontroller:
cd ~/pox
sudo ./pox.py samples.migrate

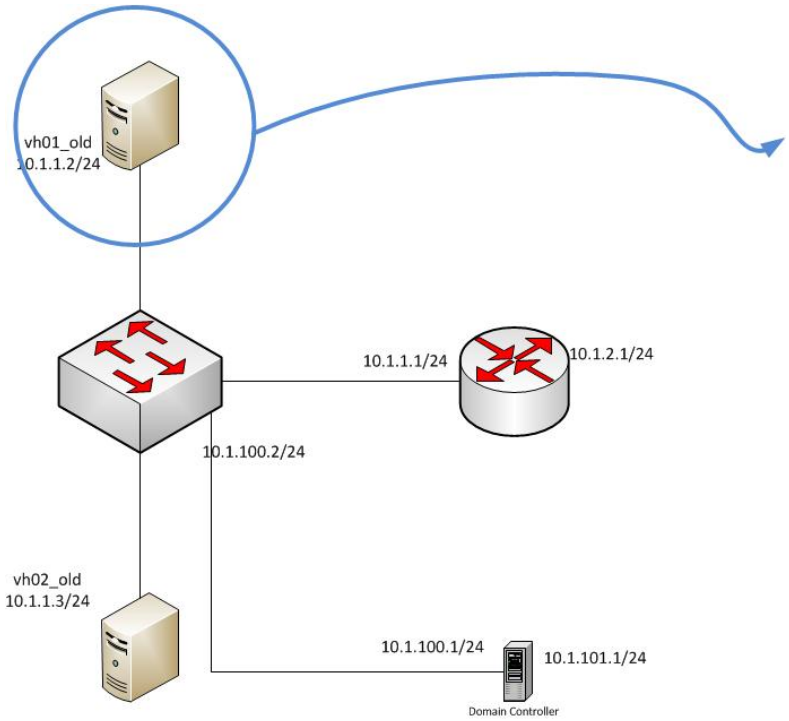
Scenarios, Experiments, and Results

The base network is depicted on the left of the router in the following figure. To migrate the network suggested by a “Tree” topology [2] we can migrate the switch and two virtual hosts together with migrate.sh . One must note that the virtual hosts must DHCP request to function on the new network. The same is true for the OpenvSwitch(vh03_new) on the OFP management network.

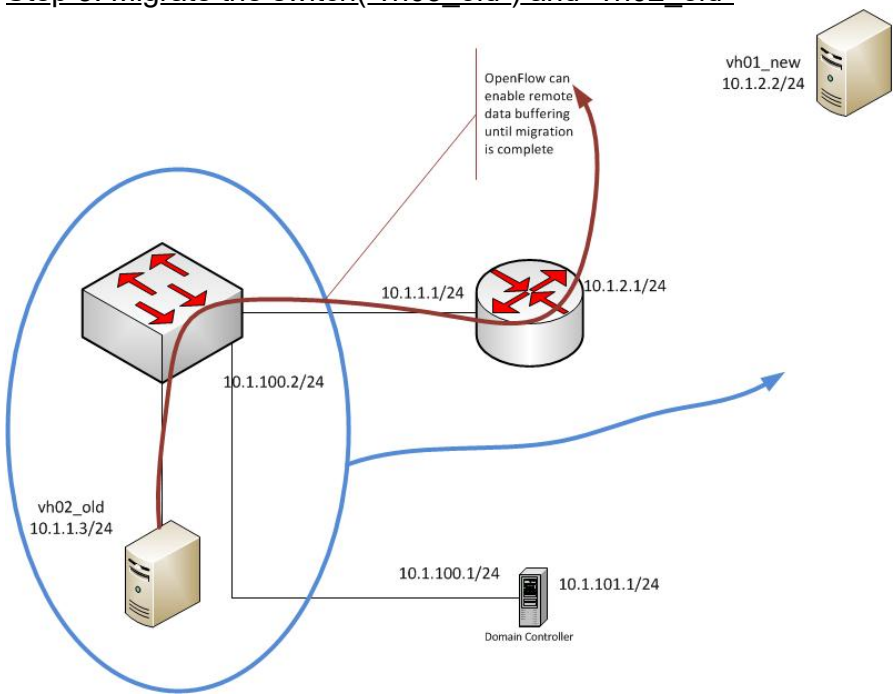


Another scenario to consider is when one VH leaves the network in advance, but still needs to acquire the data sent. One can modify the flow to forward data to a remote location where the packet may be buffered until the VH is ready to receive it. This is demonstrated in the demo by generating traffic from vh02 destined for vh01. In the case that vh01 leaves the network a flow can be started(on the switch) from vh02 to a remote destination for buffering.

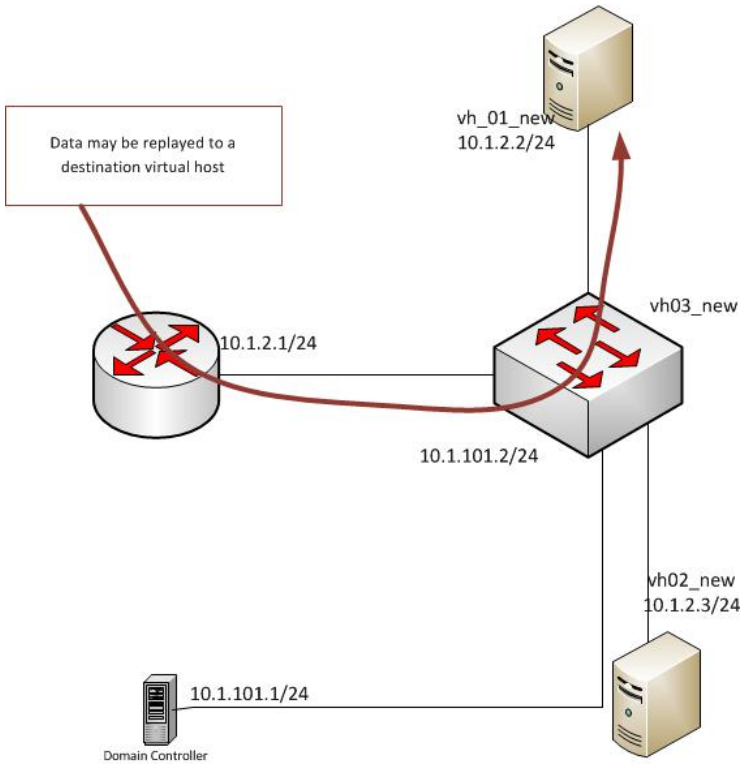
Step 1: Migrate only “vh01_old”



Step 2: Start the data flow on switch to send the data to remote location for buffering.
Step 3: Migrate the switch(“vh03_old”) and “vh02_old”



Step 4: Once vh01_new is available the remote buffer may “replay” the buffered packets to vh01_new



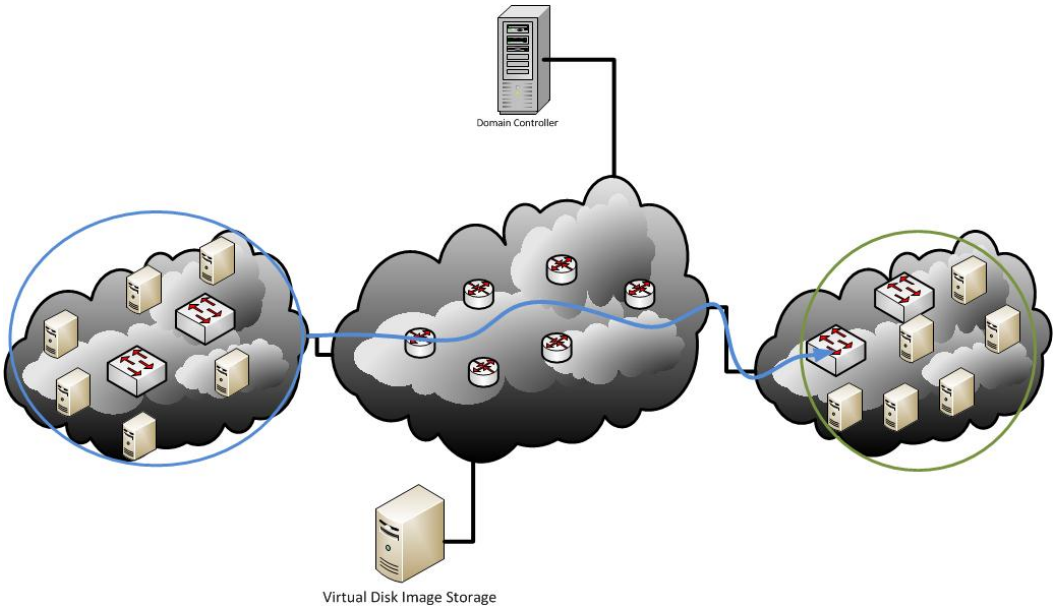
Conclusions

After reading the research papers and implementing the project one will find OpenFlow to be of use to greatly reduce network management and complexity. In addition to these reductions benchmark testing has proven OpenFlow to be an effective tool whether it be network migration or any other administrative changes. By reducing the complexity of management and increasing controls on flows power consumption can be reduced. OpenFlow is a great compliment to current networks and should drive evolution of NGN architecture. While Cisco [11]denounces SDN (perhaps due to a threat to its current business model), the evolution will be inevitable with market demand. According to Kyle Forster(co-founder) of Big Switch there is a need for control on levels between ASIC speeds and human interaction speeds [12]. Openflow fills in this gap to add more logic and make the network more intelligent. With this in mind it is a great addition for network migration even on a larger scale.

Possible Future Work

The prototype is proof of concept for the SDN network migration model that can be implemented on a larger scale. The prototype model can be expanded to:
- host a vdi (iSCSI/FHS) SAN storage facility at a remote location
- control the network from a remote location
- migrate an SDN network to a truly remote location

In SIP/IMS routing there is a 3GPP/ETSI concept called RACF (Resource Allocation Control Function). A dynamic RACF can be implemented over the NGN MPLS core network via OpenFlow.



References

[1] Pisa, Pedro, et al. "Openflow and Xen-based virtual network migration."Communications: Wireless in Developing Countries and Networks of the Future(2010): 170-181.
[2] Ghorbani, Soudeh, et al. "Transparent, Live Migration of a Software-Defined Network."
[3] Clark, Christopher, et al. "Live migration of virtual machines." Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2. USENIX Association, 2005.
[4] Boughzala, Bochra, et al. "OpenFlow supporting inter-domain virtual machine migration." Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on. IEEE, 2011.
[5] Pu, Yan, Yilong Deng, and Aki Nakao. "Cloud rack: Enhanced virtual topology migration approach with open vswitch." Information Networking (ICOIN), 2011 International Conference on. IEEE, 2011.
[6] Arora, Dushyant, and Diego Perez Botero. "Live Migration of an Entire Software-Defined Network."
[7] End-user-documentation-Oracle VM Virtualbox. ORACLE. 10 December 2012. https://www.virtualbox.org/manual/UserManual.html
[8] OpenFlow Tutorial - OpenFlow Wiki. Stanford University. 10 December 2012. http://www.openflow.org/wk/index.php/OpenFlow_Tutorial
[9] POX Wiki - Open Networking Lab - Confluence. Stanford University. 11 December 2012. https://openflow.stanford.edu/display/ONL/POX+Wiki
[10] MininetWalkthrough<OpenFlow<Foswiki. 10 December 2012. http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/MininetWalkthrough
[11] Cisco Systems, Inc. 11 December 2012. http://www.cisco.com/
[12] InternetNewsBlog. (2012, December 11).Big Switch Co-Founder Kyle Forster Talks OpenFlow. Retrieved from http://www.youtube.com/watch?v=LcnasInEE4o

Appendix

OpenFlow

"""
This component is used to control the flows during a network migration in a test network.

```
It acts as a

Authored by: Nikhil Khatu for the NCSU ECE570 Course.
"""

from pox.core import core
from pox.lib.addresses import IPAddr, EthAddr
import pox.openflow.libopenflow_01 as of


log = core.getLogger()



class migrate (object):

  def __init__(self, connection):

    self.connection = connection
    connection.addListeners(self)

    #(keys are MACs, values are ports).
        self.mactable = {}

  def send_packet (self, buffer_id, raw_data, out_port, in_port):
    msg = of.ofp_packet_out()
    msg.in_port = in_port

    if buffer_id != -1 and buffer_id is not None:
        msg.buffer_id = buffer_id
    elif raw_data is None: return
    else: msg.data = raw_data

    action = of.ofp_action_output(port=out_port)
    msg.actions.append(action)

    self.connection.send(msg)


  def _handle_PacketIn (self, event):

    parsed_data_packet = event.parsed
    openflow_packet = event.ofp

    self.mactable[parsed_data_packet.src] = openflow_packet.in_port
    if parsed_data_packet.dst not in self.mactable: self.send_packet(openflow_packet.buffer_id, openflow_packet.data,
of.OFPP_FLOOD, openflow_packet.in_port)

    elif parsed_data_packet.dst == EthAddr("08:00:27:2d:2f:e2"):
        #print 'Out port1: ', self.mactable[parsed_data_packet.dst], 'Destination address: ', parsed_data_packet.dst
        msg = of.ofp_flow_mod()
        msg.command = of.OFPFC_MODIFY_STRICT
        msg.match.dl_dst = EthAddr("08:00:27:2d:2f:e2")
        msg.match.dl_src = EthAddr("08:00:27:89:cd:69")
        msg.match.nwsrc = "10.1.1.3/24"
        msg.hard_timeout = 30


        action_one = of.ofp_action_nw_addr.set_dst(IPAddr("10.5.6.34"))
        msg.actions.append(action_one)
        action_two = of.ofp_action_output(port = 3)
        msg.actions.append(action_two)

        self.connection.send(msg)

    else:
        self.send_packet(openflow_packet.buffer_id, openflow_packet.data, self.mactable[parsed_data_packet.dst],
openflow_packet.in_port)
        #print 'Out port2: ', self.mactable[parsed_data_packet.dst], 'Destination address: ', parsed_data_packet.dst




def launch ():
  def start_switch (event):
    log.debug("Controlling %s" % (event.connection,))
    migrate(event.connection)

  core.openflow.addListenerByName("ConnectionUp", start_switch)
```

Script files

create_original_vh01.sh :

```
#-------------Register vh01_old ------------------
VBoxManage createvm --name "vh01_old Clone_test" --register

#------------- Adjust the VM settings ----------------------
VBoxManage modifyvm "vh01_old Clone_test" --memory 2048 --ostype Ubuntu_64 --acpi on --vram 12 --ioapic on --pae
off --rtcuseutc on --audiocontroller ac97
VBoxManage modifyvm "vh01_old Clone_test" --nic1 intnet --macaddress1 0800272d2fe2 --cableconnected1 on --
intnet1 "net01" --nictype1 Am79C973

#------------ Enable promiscuous mode on NIC--------------------
VBoxManage setextradata "vh01_old Clone_test" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/IfPolicyPromisc" "allow-
all"

#------------------Attach(and create if needed) storage to VM-------------
#VBoxManage createhd --filename "vh01_old Clone_test.vdi" --size 10000
VBoxManage storagectl "vh01_old Clone_test" --name "IDE Controller" --add ide --controller PIIX4
VBoxManage storageattach "vh01_old Clone_test" --storagectl "IDE Controller" --port 1 --device 0 --type dvddrive --
medium emptydrive
VBoxManage storagectl "vh01_old Clone_test" --name "SATA Controller" --add sata --sataportcount 1 --bootable on
VBoxManage storageattach "vh01_old Clone_test" --storagectl "SATA Controller" --port 0 --device 0 --type hdd --medium
/home/ngkhatu/Desktop/vh01_old\ Clone.vdi
```

create_original_vh02.sh:

```
#-------------Register vh02_old ------------------
VBoxManage createvm --name "vh02_old Clone_test" --register

#------------- Adjust the VM settings ----------------------
VBoxManage modifyvm "vh02_old Clone_test" --memory 2048 --ostype Ubuntu_64 --acpi on --vram 12 --ioapic on --pae
off --rtcuseutc on --audiocontroller ac97
VBoxManage modifyvm "vh02_old Clone_test" --nic1 intnet --macaddress1 08002789cd69 --cableconnected1 on --
intnet1 "net02" --nictype1 Am79C973

#------------ Enable promiscuous mode on NIC--------------------
VBoxManage setextradata "vh02_old Clone_test" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/IfPolicyPromisc" "allow-
all"

#------------------Attach(and create if needed) storage to VM-------------
#VBoxManage createhd --filename "vh02_old Clone_test.vdi" --size 10000
VBoxManage storagectl "vh02_old Clone_test" --name "IDE Controller" --add ide --controller PIIX4
VBoxManage storageattach "vh02_old Clone_test" --storagectl "IDE Controller" --port 1 --device 0 --type dvddrive --
medium emptydrive
VBoxManage storagectl "vh02_old Clone_test" --name "SATA Controller" --add sata --sataportcount 1 --bootable on
VBoxManage storageattach "vh02_old Clone_test" --storagectl "SATA Controller" --port 0 --device 0 --type hdd --medium
/home/ngkhatu/Desktop/vh02_old\ Clone.vdi
```

create_original_vh03.sh:

```
#-------------Register vh03_old ------------------
VBoxManage createvm --name "vh03_old Clone_test" --register

#------------- Adjust the VM settings ----------------------
VBoxManage modifyvm "vh03_old Clone_test" --memory 2048 --ostype Ubuntu_64 --acpi on --vram 12 --ioapic on --pae
off --rtcuseutc on --audiocontroller ac97
#------------NIC settings-------------------------
VBoxManage modifyvm "vh03_old Clone_test" --nic1 intnet --macaddress1 auto --cableconnected1 on --intnet1 "net01" -
-nictype1 Am79C973
VBoxManage modifyvm "vh03_old Clone_test" --nic2 intnet --macaddress2 auto --cableconnected2 on --intnet2 "net02" -
-nictype2 Am79C973
VBoxManage modifyvm "vh03_old Clone_test" --nic3 intnet --macaddress3 auto --cableconnected3 on --intnet3 "net03" -
-nictype3 Am79C973
VBoxManage modifyvm "vh03_old Clone_test" --nic4 intnet --macaddress4 080027bbd531 --cableconnected4 on --
intnet4 "netof01" --nictype4 Am79C973
#------------ Enable promiscuous mode on NIC--------------------
VBoxManage setextradata "vh03_old Clone_test" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/IfPolicyPromisc" "allow-
all"
VBoxManage setextradata "vh03_old Clone_test" "VBoxInternal/Devices/pcnet/1/LUN#0/Config/IfPolicyPromisc" "allow-
all"
VBoxManage setextradata "vh03_old Clone_test" "VBoxInternal/Devices/pcnet/2/LUN#0/Config/IfPolicyPromisc" "allow-
all"

#------------------Attach(and create if needed) storage to VM-------------
#VBoxManage createhd --filename "vh03_old Clone_test.vdi" --size 10000
VBoxManage storagectl "vh03_old Clone_test" --name "IDE Controller" --add ide --controller PIIX4
VBoxManage storageattach "vh03_old Clone_test" --storagectl "IDE Controller" --port 1 --device 0 --type dvddrive --
medium emptydrive
VBoxManage storagectl "vh03_old Clone_test" --name "SATA Controller" --add sata --sataportcount 1 --bootable on
VBoxManage storageattach "vh03_old Clone_test" --storagectl "SATA Controller" --port 0 --device 0 --type hdd --medium
/home/ngkhatu/Desktop/vh03_old\ Clone.vdi
```

make_vh01_clone.sh:

```
#--------Register vh01_new----------
VBoxManage createvm --name "vh01_new Clone_test" --register

#-------------Adjust the VM settings (exactly the same as old)-----------------
VBoxManage modifyvm "vh01_new Clone_test" --teleporter on --teleporterport "6000" --teleporterpassword csc570
VBoxManage modifyvm "vh01_new Clone_test" --memory 2048 --ostype Ubuntu_64 --acpi on --vram 12 --ioapic on --
pae off --rtcuseutc on --audiocontroller ac97
#--------------NIC settings------------------
VBoxManage modifyvm "vh01_new Clone_test" --nic1 intnet --macaddress1 0800272d2fe2 --cableconnected1 on --
intnet1 "net04" --nictype1 Am79C973
#----------------Enable promiscuous mode on NIC---------------
VBoxManage setextradata "vh01_new Clone_test" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/IfPolicyPromisc"
"allow-all"

#-----------------Attach storage to VM----------------------------
VBoxManage storagectl "vh01_new Clone_test" --name "IDE Controller" --add ide --controller PIIX4
VBoxManage storageattach "vh01_new Clone_test" --storagectl "IDE Controller" --port 1 --device 0 --type dvddrive --
medium emptydrive
VBoxManage storagectl "vh01_new Clone_test" --name "SATA Controller" --add sata --sataportcount 1 --bootable on
VBoxManage storageattach "vh01_new Clone_test" --storagectl "SATA Controller" --port 0 --device 0 --type hdd --
medium /home/ngkhatu/Desktop/vh01_old\ Clone.vdi
```

make_vh02_clone.sh:

```
#--------Register vh02_new----------
VBoxManage createvm --name "vh02_new Clone_test" --register

#-------------Adjust the VM settings (exactly the same as old)-----------------
VBoxManage modifyvm "vh02_new Clone_test" --teleporter on --teleporterport "6001" --teleporterpassword csc570
VBoxManage modifyvm "vh02_new Clone_test" --memory 2048 --ostype Ubuntu_64 --acpi on --vram 12 --ioapic on --
pae off --rtcuseutc on --audiocontroller ac97
#--------------NIC settings------------------
VBoxManage modifyvm "vh02_new Clone_test" --nic1 intnet --macaddress1 08002789cd69 --cableconnected1 on --
intnet1 "net05" --nictype1 Am79C973

#----------------Enable promiscuous mode on NIC---------------
VBoxManage setextradata "vh02_new Clone_test" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/IfPolicyPromisc"
"allow-all"

#-----------------Attach storage to VM----------------------------
VBoxManage storagectl "vh02_new Clone_test" --name "IDE Controller" --add ide --controller PIIX4
VBoxManage storageattach "vh02_new Clone_test" --storagectl "IDE Controller" --port 1 --device 0 --type dvddrive --
medium emptydrive
VBoxManage storagectl "vh02_new Clone_test" --name "SATA Controller" --add sata --sataportcount 1 --bootable on
VBoxManage storageattach "vh02_new Clone_test" --storagectl "SATA Controller" --port 0 --device 0 --type hdd --
medium /home/ngkhatu/Desktop/vh02_old\ Clone.vdi
```

make_vh03_clone.sh:

```
#--------Register vh03_new----------
VBoxManage createvm --name "vh03_new Clone_test" --register

#-------------Adjust the VM settings (exactly the same as old)-----------------
VBoxManage modifyvm "vh03_new Clone_test" --teleporter on --teleporterport "6002" --teleporterpassword csc570
VBoxManage modifyvm "vh03_new Clone_test" --memory 2048 --ostype Ubuntu_64 --acpi on --vram 12 --ioapic on --
pae off --rtcuseutc on --audiocontroller ac97
#--------------NIC settings------------------
VBoxManage modifyvm "vh03_new Clone_test" --nic1 intnet --macaddress1 auto --cableconnected1 on --intnet1 "net04"
--nictype1 Am79C973
VBoxManage modifyvm "vh03_new Clone_test" --nic2 intnet --macaddress2 auto --cableconnected2 on --intnet2 "net05"
--nictype2 Am79C973
VBoxManage modifyvm "vh03_new Clone_test" --nic3 intnet --macaddress3 auto --cableconnected3 on --intnet3 "net06"
--nictype3 Am79C973
VBoxManage modifyvm "vh03_new Clone_test" --nic4 intnet --macaddress4 080027bbd531 --cableconnected4 on --
intnet4 "netof02" --nictype4 Am79C973
#----------------Enable promiscuous mode on NIC---------------
VBoxManage setextradata "vh03_new Clone_test" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/IfPolicyPromisc"
"allow-all"
VBoxManage setextradata "vh03_new Clone_test" "VBoxInternal/Devices/pcnet/1/LUN#0/Config/IfPolicyPromisc"
"allow-all"
VBoxManage setextradata "vh03_new Clone_test" "VBoxInternal/Devices/pcnet/2/LUN#0/Config/IfPolicyPromisc"
"allow-all"

#-----------------Attach storage to VM----------------------------
VBoxManage storagectl "vh03_new Clone_test" --name "IDE Controller" --add ide --controller PIIX4
VBoxManage storageattach "vh03_new Clone_test" --storagectl "IDE Controller" --port 1 --device 0 --type dvddrive --
medium emptydrive
VBoxManage storagectl "vh03_new Clone_test" --name "SATA Controller" --add sata --sataportcount 1 --bootable on
VBoxManage storageattach "vh03_new Clone_test" --storagectl "SATA Controller" --port 0 --device 0 --type hdd --
medium /home/ngkhatu/Desktop/vh03_old\ Clone.vdi
```

remove_migrate.sh:

```
VBoxManage unregistervm "vh01_new Clone"
VBoxManage unregistervm "vh02_new Clone"
VBoxManage unregistervm "vh03_new Clone"
```

start_base.sh:

```
VBoxManage startvm "ofcontroller Clone"
VBoxManage startvm "router Clone"
sleep 1
VBoxManage startvm "vh03_old Clone_test"
sleep 5
VBoxManage startvm "vh01_old Clone_test"
VBoxManage startvm "vh02_old Clone_test"
```

power_off_base.sh:

```
VBoxManage controlvm "vh01_old Clone_test" poweroff
VBoxManage controlvm "vh02_old Clone_test" poweroff
VBoxManage controlvm "vh03_old Clone_test" poweroff
VBoxManage controlvm "ofcontroller Clone" poweroff
VBoxManage controlvm "router Clone" poweroff
```

migrate.sh:

```
VBoxManage startvm "vh03_new Clone_test" &
sleep 1
VBoxManage controlvm "vh03_old Clone_test" teleport --host "localhost" --port "6002" --password csc570
VBoxManage startvm "vh01_new Clone_test" &
sleep 1
VBoxManage controlvm "vh01_old Clone_test" teleport --host "localhost" --port "6000" --password csc570
VBoxManage startvm "vh02_new Clone_test" &
sleep 1
VBoxManage controlvm "vh02_old Clone_test" teleport --host "localhost" --port "6001" --password csc570
```

power_off_migrate.sh:

```
VBoxManage controlvm "vh01_new Clone_test" poweroff
VBoxManage controlvm "vh02_new Clone_test" poweroff
VBoxManage controlvm "vh03_new Clone_test" poweroff
VBoxManage controlvm "ofcontroller Clone" poweroff
VBoxManage controlvm "router Clone" poweroff
```