Self-Provisioning of Network Infrastructure for Server Landings in Complex Data Center Environments

Patrick Apel
Automated Network Management
Intel Corporation
Folsom, California, USA
doug.apel@intel.com

Greg Botts, Alex Fullam
Enterprise DataCenter Network Operations
Intel Corporation
Rio Rancho, New Mexico, USA
greg.botts@intel.com

***Abstract*—Overview of a network provisioning system which dynamically determines the necessary path, finds free ports, and configures the vlan trunking and spanning trees using fuzzy logic and dynamic web service composition.**

Keywords—network, automation, web services, configuration, server, provision

I. INTRODUCTION

We will provide a short summary of a current network management automation project being used internally at Intel. We built an application which integrates our existing information and tools, in order to obtain dramatic improvements in our operational responsiveness for simple and complex network provisioning in support of server landings for our public and private cloud offerings.

II. MOTIVATION AND GOALS

Intel has about 80 datacenters worldwide hosting hundreds of thousands of servers, with more than 65% of that virtualized. Our enterprise application DMZ (demilitarized zone) in particular has both high demand for new landings, and is one of our most highly secure and complicated network environments. It is spread across 4 geographic locations, and consists of thousands of VLANs and PVLANs (Private VLANs), and subnets, with multiple network equipment vendors and multiple hardware/software combinations. This complexity has made it increasingly difficult to configure the network safely and securely, such that network path provisioning has become a substantial bottleneck to overall turnaround time for server landing requests.

Other data centers are typically for dedicated purposes such as Compute farms for engineering/design purposes, and the network complexity is significantly lower in these data centers. Key motivating factors driving the development of this tool were the following:

A. Landing new servers, virtual or physical, in our enterprise data centers is imperative for elastic cloud computing
   1. 37% annual growth in landing requests
   2. 500+ server landings per year in our application DMZ alone

B. Every network interface on each server requires network path provisioning
   1. Requires intricate knowledge and precise implementations for safe, secure, and stable operation
   2. Each server typically has between 3 and 16 interfaces, and each one requires separate network path provisioning

C. The network provisioning piece was simply taking too long to accommodate rapid server landings
   1. Laborious manual processes and data lookups due to thousands of VLANs, Private VLANs, and subnets to choose from
   2. Each application DMZ typically has between 125 and 250 VLANs

We sought to create a Network Automation tool that empowered the server landing team to Verify, Reserve, Assign, Configure and Test network connectivity for new landings. This tool needed to meet these high-level requirements:

A. Decrease the need for network operations involvement by reducing the workload for complex requests, and eliminating the workload for simple requests

B. Decrease the overall landing cycle time,

C. Reduce change-related incidents and decrease rework/escalation costs,

D. Handle both on-demand landings and future landing reservations

E. Maintain change audit log using network device configuration management engine

The existing service-level agreement allows for 10 days for the network provisioning portion of server landings. The actual configuration effort of course takes far less time, but 10 days was required (and frequently breached) as a result of the complex research, analysis, consultations, and decision-making required to determine precisely how each network interface on a server should be configured at the switch-side, in order to securely land that particular application. With each server having between 3 and 16 network interfaces, trying to

978-3-901882-50-0

determine which VLANs and PVLANs and switches and ports could be quite daunting.

Our target was that 10+ day window. If we could significantly decrease that time, the overall server landing time could be significantly reduced. The time to build the actual server has already dropped to less than 45 minutes, so the 10 day network implementation window was being increasingly seen as the real roadblock after all the human request and approval loops were complete.

We took on the challenge of trying to automate the network path provisioning, specifically in this complex application DMZ environment because it had the most urgent need. We also expected that if we could make a modular and generic application which could handle the inherent complexity of this environment, that we would also have a viable solution for the rest of our internal datacenter locations.

As a result, while all of our designs accommodate the specific requirements of our enterprise data centers, they also have no specific dependencies on the application DMZ or Intel data centers in particular. Our objectives are likewise generic. Many public and private datacenters have similar scenarios. The subnetworks and VLANs have grown organically and not necessarily logically over the years, with new ones being slotted into unpredictable spots. As a result, existing SDN (Software-Defined Networking) such as OpenStack's "nova-manage network create" [1], which works well in blank-slate clouds, tend to be poorly suited to the contained chaos of existing datacenters. Newer methods such as OpenStack Quantum can provide an abstraction layer on top of existing complex datacenters, but introduce potential performance, stability, and security penalties with the additional network complexity of tunnelling and fabric overlays. Determining the correct business logic is the critical piece, even if the SDN tools have the ability to fully perform the required tasks.

III. ARCHITECTURE AND INTERFACE

Intel's enterprise application DMZ environment is exceptionally complicated. It provides an enclaved/secure environment for external access such as hosting services, semiconductor foundry partner applications, and multi-company business-to-business gateways. It also provides internal secure and non-secure application landing zones for our large private cloud. In order to safely and securely perform the network provisioning, our system must navigate complex decision trees using fuzzy logic to choose the optimal solution, while adhering to specific rules to prevent accidentally connecting any purposefully-separated networks. Additional details on the fuzzy logic implementation are provided futher in this section.

The application consists of these major components:

A. In-house developed PHP web front-end

B. In-house developed fuzzy logic decision trees

C. SQL data store as part of larger Federated CMDB

D. Multivendor SOAP APIs

E. Javascript/AJAX for real-time queries and data validation

F. Commercial network device configuration management engine

1. In-house developed device configuration scripts for gathering information and making changes to switches

   a. free port detection and multi-protocol neighbor relationship detection

   b. access port, channel port, and uplink port configuration

   c. VLAN and PVLAN creation and trunking

G. Commercial IP Address management (IPAM) engine

1. DNS, subnet, and VLAN/PVLAN mapping

We rapidly prototyped a web-based front-end with PHP to develop this solution. The PHP application acts as the fuzzy logic decision engine, taking in various inputs from our federated CMDB SQL sources, and making API queries against our existing commercial IPAM engine and our existing network configuration engine to build Bayesian models with probabilistic functions.

After the proper decisions have been reached about what network equipment to configure, and how to properly configure them, the PHP application initiates a sequence of API calls with dynamic web service composition [5],[6] to the network configuration engine to invoke jobs against those network devices. The jobs refer to existing pre-defined "Standard Change" modular/atomic action scripts which we have developed over the past 2 years.

The automated scripts are written in XML using a semi-proprietary schema which is similar to Expect script: prompt, command, response, plus pattern-recognition/capture and conditional flow-of-control logic. These actions are very atomic tasks, such as "Configure an Access Port", and take a variable number of runtime parameters. Each script dynamically detects the target device hardware/software platform (which translates into Cisco IOS/CatOS/NXOS, Extreme XOS, and HP ProCurves in our environment) and runs the correct sequence of commands for that platform. These actions are also designed for portability, such that if we change our network configuration engine (say to Puppet in the future), we should be able to translate the code with a minimum of effort. When we initiated this project in late 2011, there was still incomplete support within Quantum, Nova, and Puppet with regard to the network equipment we needed to support, and incomplete feature sets for controlling advanced networking parameters such as multiple private vlans, proprietary spanning trees, and port security parameters [2].

The web form inputs for this application are limited to only the "server"-centric information which the intended audience (server owners/technicians) are already going to know:

- The site and datacenter location
- the row and rack of the server

- the name and request ID of the application
- The number of server NICs required
- The IP addresses and DNS names of those NICs
- The role/purpose and speed of those NICs

In real-time, we then determine the access-layer switches which service that particular server landing location, and which ports are currently available on those switches at the requested speeds. We implemented fuzzy logic algorithms [3] to determine which ports should be deemed the best candidates of "currently available". This is a probable value, modeled on our network operations personnel real-world decision-making process to determine how likely it is that any given switch port is free to be re-used. This evaluates multiple factors such as operational state, adminstrative state, current vlan assignment, port name/alias patterns, elapsed time since last traffic, maximum available speed, port media type, parent chassis/linecard models, and the likelihood of pre-patched cabling in certain rack locations.

If sufficient switch ports are calculated to be available at the requested location, then we perform another round of fuzzy calculations to determine the optimal mix of switch ports to satisfy the required server interfaces, taking into account factors such as load-balancing production server interfaces across multiple network switches for fault-tolerance with multi-variate round-robining, and optimizing resource usage by using the lowest-performance/lowest-cost switch port to satisfy each specific server network interface. These fuzzy decisions are based on the ICL (Independent Choice Logic) model [4] by extending our acyclic decision tree with a probability distribution for the available choices. The ICL model is particularly appropriate because it provides an organized way to express our frequent context-specific conditional relationships, such as "*a may be independent of c when b is false but be dependent when b is true*" [4].

From the supplied IP addresses of the server network interfaces, we do a lookup within our IP address management system to determine which VLANs/PVLANs best correspond to those subnets. We then check if those VLANs/PVLANs are already present on the distribution (layer-3) switches and targeted access/edge switches, or if they will need to be created and/or trunked to extend each required VLAN down to that particular access switch port.

When the submit button is pushed, the application begins a planning and matchmaking sequence to determine the correct sequences of SOAP API calls and then initiates those API calls asynchronously or synchronously based on whether any computed tasks are dependent upon predecessor tasks [5]. Those API calls are made to our network configuration engine to launch the modular tasks in the required order in order to safely implement the desired changes at the access and distribution layers as necessary [6]. By composing our Web Services calls into automated sequences, we ensure robust and reliable delivery of the intended service. (The reader is directed to [7] for an excellent overview of various automated web service composition methods.) Lastly, a cabling request is automatically generated for submission to our datacenter cabling contractors to perform any required patch-cabling between the server interfaces and the configured switch ports.

IV. Challenges and Key Dependencies

Because we adopted this automated approach on top of our existing legacy manual processes, we had to overcome a large number of dependencies. While we do use fuzzy logic, we cannot make decisions based on human intuition and guesswork. A key design win was the creation of a generic metadata repository built with automatically detected relationships, removing almost all the "human glue". Prior to the introduction of this system, many of the decisions were dependent on tribal knowledge. Establishing the metadata repository (and clear ownership for its maintenance) allowed us to definitively answer questions such as:

- Which switches provide service for which rows/racks?
- Which switches provide internal vs external networks vs both?
- Which vlans map to which subnets?
- Which vlans map to private vlans, and what types of private vlans?
- Which switchports are uplinks/downlinks to other switches/routers?

We had an existing library of well-proven automation scripts which we had previously written for use with our commerical network configuration engine. We wanted to leverage these already-approved modular scripts by chaining them together to perform complex operations composed of many simple building blocks [6].

A simple 3-NIC server landing generates ~12 discrete automated tasks, assuming all the VLANs are pre-existing. In contrast, a more complex 8-NIC server landing generates ~60 discrete automated tasks, assuming all the VLANs need to be created and extended.

V. Results

We have demonstrated that we can implement automated network port provisioning to satisfy server landing requests in real-time, in a highly complex network environment, and do so in a safe and efficient manner. We recognize that this is likely a stop-gap measure to faciliate our rapid public and private cloud expansion while we explore the scalability and robustness of emerging Software-Defined Networks methods. For now, this is our practical implementation of existing proven technologies to satisfy our immediate goals: speed and stability. Our solution delivered these results:

- Self-service real-time network provisioning
  - 10-Day SLA (frequently breached) reduced to 5 minutes on-demand in real-time.
  - Reduces burden on network operations staff (12-15%)
  - Removes the (real and perceived) roadblock of "network"

- Increases agility of cloud expansion

- Reliable, consistent, and trackable changes
  - Full audit trail, every step logged for chain of custody
  - Fewer configuration mistakes and omissions
  - Optimizes port/switch resource usage and fault-tolerance

- Reservation system for future landing requests
  - Allows ports to be reserved now for a future landing
    - When the server does land, it will prefer that previously reserved port
  - Automatic age-out of stale reservations
  - Reports and dashboards to show total, in-use, and reserved ports for short- and long-term capacity planning for each DC down to the row and rack level.

VI. KEY FEATURES AND KEY LEARNINGS

This system is an integration of our existing data and tools, in order to produce end-to-end network path provisioning in support of physical and virtual server landings. In addition to integrating the data and tools, we added a fuzzy logic engine to build web service chains with dynamic business logic, using decision trees based on our human network operator processes.

A. *Key Features*

1. This is an integrated system which leverages our federated CMDB for data-driven decision-making
2. Web-based front-end form which asks only "server"-related questions to make the proper network choices
3. API calls to IP Address Management system to determine proper VLANs/PVLANs to be associated with hosts
4. API calls to metadata repository to determine which datacenter rows/racks are associated with which access switches
5. API calls to network device configuration management system to perform config changes to distribution and access switches
6. Fuzzy logic algorithms using Independent Choice Logic structures to:
   a. make "best fit" decisions about switches and ports,
   b. prevent inter-connecting segregated networks

B. *Key Learnings*

1. The most complicated aspects of this automation effort were related to identifying and codifying the tribal knowledge
2. Our prior art in defining automated "Standard Change" tasks for Cisco, HP, and Extreme devices allowed us to re-use those existing function libraries with minor modifications
3. Provisioning the network for server landings is deceptively complicated in enterprise environments, and can be composed of dozens of discrete network tasks

VII. CONCLUSIONS

Our solution provides a highly flexible automated network path provisioning system which dynamically supports multiple network device types, multiple network designs, multiple server network interfaces, and multiple network VLANs and Private VLANs. It promotes on-demand self-service to facilitate cloud explansion by our data center hosting server owners with little or no network knowledge. Most importantly, it delivers consistent, rapid, and predictable results despite the underlying complexity, and it does so while being easy for non-network personnel to use. This in turn reduces the workload on the network personnel, freeing them up for more value-added projects.

We will continue to explore wide-spread generic applicability of this solution to other Intel datacenter environments. We are also investigating moving from our current proprietary network configuration engine to Puppet or other open source solutions.

ACKOWLEDGMENTS

The authors wish to thank Radia Perlman, Intel Fellow, for several in-depth interviews regarding Software Defined Networking implementations.

REFERENCES

[1] OpenStack Compute Administration Manual, Folsom 2012.2 ed., Openstack Foundation, 2012 (2013, Jan 10). [Online]. Available: http://docs.openstack.org/trunk/openstack-compute/admin/content/configuring-vlan-networking.html

[2] B. Figureau. (2013, Jan 10). *Puppet Network Device Management* [Online]. Available: http://puppetlabs.com/blog/puppet-network-device-management/

[3] R. Behlohlavek, V.Vychodil, "Alegbras with Fuzzy Equalties," in Fuzzy Sets and Systems, vol. 157 issue 2, Jan 2006, pp. 161-201.

[4] D. Poole, "The Independent Choice Logic and Beyond," in Lecture Notes in Computer Science, vol 4911, Probabilistic Inductive Logic Programming, Springer 2008, pp 222-243.

[5] T. Klie, F. Gebhard, S. Fischer, "Towards Automatic Composition of Network Management Web Services," 10th IFIP/IEEE Integrated Network Management 2007 Int. Symp., May 2007. pp 769-772.

[6] I. Weber, "Task Composition," in Lecture Notes in Business Information Processing, vol 40, Semantic Methods for Execution-level Business Modeling, Springer 2009, pp. 149-200.

[7] P. Bartalos, M. Bielikova. "Automatic Dynamic Web Service Composition: A Survey and Problem Formalization," Computing and Informatics 2011, vol 30, no. 4, pp. 793-827.