Agent based auto-configuration of OSPF networks

Visa Holopainen
visa.holopainen@tkk.fi
Problem

- About 30% - 50% of network outages are caused by configuration error (The Yankee Group's 2003 query)

- It would be nice to automate manual error-prone configuration tasks
Previous work in auto-configuration

Many high-level proposals
WAN: routers

Many working systems
LAN: workstations/modems/switches
Our work

A working system

WAN: routers
What is it?

• The main idea in our system is following:
  – A person with very limited or no computer skills can take several commodity PCs, connect their Network Interface Cards (NICs), plug a specifically configured laptop to one of those PCs, and after a while the network will be a fully functional OSPF network.
Problem

• At first there’s no connectivity to routers (unless a separate management LAN is set up)

• How about auto-configuration using CD/USB?
  – possible but may be logistically complicated if routers are in different physical locations
Our solution

- "Flood" routing protocol to the network -> incrementally increase connectivity range until all routers have been configured
auto-configuration process (1)

Autoconfig package contains:
1. agent Perl-script
2. routing software tarball
3. a file that contains IP addresses of interfaces that have already been configured to run OSPF (the master script maintains this file)
4. a policy file
auto-configuration process (2)
Policy file

- Plain and simple
- Suitable default values in most parts
- networking novice should be able to fill in suitable values where needed

```plaintext
default-internet-gateway-IP-address="10.0.0.1"

MD5-authentication = "yes"

permit-router-remote-configuration = "no"

link-cost-assignment-policy = "I"
# (inverse-of-link-capacity (I)

load-balancing-for-equal-cost-paths = "no"
```

...
function autoconfiguration_master {

// initializations
candidate_IPs = ();
configured_IPs = ();

candidate_IPs <- enqueue (discover_IPs());

// first time there should be 1 candidate
while(candidate_IPs != empty) {
  IP = dequeue (candidate_IPs);
  if (IP exists in configured_IPs) next;
  send_configured_IPs_to (IP);
  send_policy_file_to (IP);
  send_routing_software_tarball_to (IP);
  send_configuration_agent_to (IP);
  if (first_time)
    send_perl_library (IP);
  start_configuration_agent_at (IP);
  configured_IPs <- enqueue
    (feedback_from_agent);
  candidate_IPs <- enqueue
    (feedback_from_agent);
}
}
function autoconfiguration_agent {

    // initializations
    stubs = ();
    own_IPs = ();
    candidate_IPs = ();
    own_IPs = discover_own_IPs();
    interfaces = discover_own_interfaces();
    configured_IPs <- read_from_file();

    for each interface in interfaces
    {
        for each IP in range of interface
        {
            if IP exists in configured_IPs
            next IP;
            if (connect_success(IP))
            {
                candidate_IPs <- IP
                prepare_candidate (IP);
            }
            passive_interfaces <- interface;
        }
        send_to_master (own_IPs);
        send_to_master (candidate_IPs);
        extract_configure_and_install_routing_software
        (own_IPs, passive_interfaces);
        start_routing ();
    }
}
Auto-configuration basic stuff

• Each ethernet-interface that the agent detects (using ifconfig-tool) will cause the agent to write the following line to configuration file: `network X.Y.Z.V/N area 0`. Here X.Y.Z.V/N is the IP network configured for the interface.
Policy -> Configuration

• The agent writes configuration commands based on the policy file that the master sent
  – if (policy_entry X) then write_configuration Y
• Example 1: If the agent notices that one of the interfaces on the PC it is running on has the IP address of default gateway specified in policy file, then the agent writes the following to configuration file: `default-information originate`.
Policy

->

Configuration examples

- policy:
  permit-router-remote-configuration
  "no"
  configuration:
  access-list term permit 127.0.0.1/3
  access-list term deny any
  line vty
  access-class term

- policy:
  MD5-authentication = "yes"
  configuration:
  under each interface <ifname> line:
  ip ospf authentication
  message-digest
  ip ospf message-digest-key 1 md5
  <KEY>
  under router ospf line:
  area <AREA> authentication
  message-digest
Features

• Currently in addition to the basic routing functionality:
  – **Stubnets**: if there is no reply from an interface it is configured to be passive
  – **Broadcast networks** = switches/hubs are allowed
  – **Default Internet gateway** (not tested)
  – **ECMP load balancing** (not tested)
  – `<your suggestion>`
  – other protocols like BGP (maybe in the future)
Performance (1)

<table>
<thead>
<tr>
<th>Event</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test started</td>
<td>0</td>
</tr>
<tr>
<td>Sent agent to A</td>
<td>291</td>
</tr>
<tr>
<td>A done</td>
<td>1799</td>
</tr>
<tr>
<td>Sent agent to B</td>
<td>1978</td>
</tr>
<tr>
<td>B done</td>
<td>2969</td>
</tr>
<tr>
<td>Sent agent to C</td>
<td>3147</td>
</tr>
<tr>
<td>C done</td>
<td>3788</td>
</tr>
<tr>
<td>Sent agent to D</td>
<td>3967</td>
</tr>
<tr>
<td>D done</td>
<td>4298</td>
</tr>
</tbody>
</table>
Performance (2)

<table>
<thead>
<tr>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test started</td>
<td>0 sec</td>
</tr>
<tr>
<td>Sent agent to A</td>
<td>292 sec</td>
</tr>
<tr>
<td>A done</td>
<td>2401 sec</td>
</tr>
<tr>
<td>Sent agent to B</td>
<td>2579 sec</td>
</tr>
<tr>
<td>B done</td>
<td>3844 sec</td>
</tr>
<tr>
<td>Sent agent to C</td>
<td>4022 sec</td>
</tr>
<tr>
<td>C done</td>
<td>4663 sec</td>
</tr>
<tr>
<td>Sent agent to D</td>
<td>4842 sec</td>
</tr>
<tr>
<td>D done</td>
<td>5172 sec</td>
</tr>
</tbody>
</table>

S-38.4030 Postgraduate Course on Networking Technology

04/12/2007
Execution time of agent (seconds)

<table>
<thead>
<tr>
<th>Agent</th>
<th>Test 1</th>
<th>Test 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>agent B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>agent C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>agent D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

• With Perl it is easy to build a working auto-configuration system for Linux network (however, debugging takes longer)

• Net::SSH::Expect is awfully slow
  – However, if you don’t know how to configure routers, an hour is a short time…
  – Also, using SSH instead of Telnet makes it possible to configure the network from a separate physical location
Future work: IP auto-assignment

Issue 1: How to get the PCs to send DHCP-discoveries at startup – need to use a script that does this?
Issue 2: A hack needed to run DHCP on multiple interfaces
Future work: OS auto-installation

Diagram:

- Master
- A
- B
- C

Steps:

1. Master sends "<set up OS mirror and FAI server>
2. Install OS
3. Auto-installation agent
4. B sends "<set up OS mirror and FAI server>
5. Install OS
6. Auto-installation agent
7. No installation candidates
8. No installation candidates
9. No installation candidates
10. <done>