



# **Programmatically Generating Topologies and Configurations**

**Wesley Griffin  
Ron Ostrenga**

**SPARTA, Inc.**

**2006 DETER Community Meeting**



# DNS Root System Emulation

- **Build functioning model of the DNS Root System**
- **Cannot model entire Internet**
  - Build a highly realistic scale-model of the Internet
- **Root Server “Clusters”**
  - 13, lettered A through M
  - Clusters, or nodes, consist of multiple servers
  - Located around the Internet at various points
  - Use Anycast Routing to provide reliability
- **From a client perspective**
  - Only a single Root Server is ever seen
- **In Short: Complex Routing Configuration**



# Internet Scale Model

- **Plan to build multiple, large, interconnected AS**
  - Emulate large-scale ISPs in real Internet
  - Establish a model for the global routing table
- **Smaller autonomous systems connected to “Tier-1”**
  - Emulate small-scale ISPs and large enterprises
- **By building the topology in this manner, we hope to gain better emulation of the real Internet**
- **DNS Root System will be “deployed” into model**
- **Build anycast nodes of DNS servers**
- **Place the clusters at various points on our “Internet”**
- **Highly ambitious, possible scaling problems**



# Programmatic Generation

- **Large scale model is impossible to build manually**
- **Developed a set of TCL procedures**
  - Included in experiment's NS file
  - Can programmatically build a topology
    - Core router LANs, border router interconnects, access router LANs, and customer LANs
  - Can now specify a “simple” array to describe layout of multiple, large, interconnected autonomous systems
- **Experiment topologies are now built programmatically**



# Programmatic Configuration

- **With topology built, routing configuration still required**
- **Developing a set of Perl procedures**
  - Run via `tb-set-node-start-cmd` from experiment's NS file
  - Will programmatically build router configuration
    - For every node in experiment
    - Handle both EGP and IGP configurations, AS interconnects, and customer connections
  - Will specify routing configurations as part of “simple” topology description array
- **Large-scale topologies using complex routing will be built and configured programmatically**



# Service Configuration

- **Since routing configuration is programmatically, why not allow other services to be configured as well?**
- **Perl procedures will be modular**
- **Allow modules to be created and added**
  - Will programmatically build configurations for services
  - First service will be DNS
    - Type of DNS server software
    - Nameserver configuration
  - Configuration included as part of “simple” array
- **Service configurations will be built programmatically**



# Current Progress

- **Topology generation handles one “type” of ISP**
- **3 AS topology with multiple interconnection links**
  - Includes customer LANs and DNS Anycast Node LANs
- **Uses IANA Reserved subnets for IP addresses**
  - Each AS assigned a subnet
  - Customers numbered from the “ISP” they connect to
- **Array is specified to include routing configurations**
  - Including IGP/EGP announcements, router loopback addresses, customer and border interconnections
  - Plan to support Quagga currently
- **Routing configuration currently parses array**
  - Debugging issue with `tb-set-node-start-cmd`



# Next Steps and Potential Issues

- **Finish Perl procedures so that routing works**
- **Incorporate support for the Junipers**
  - Likely role as border routers for AS interconnections
- **Write Perl procedures for service configurations**
  - Focus on BIND configuration initially
  - Add support for NSD and ANS
- **Largest Potential Issue: Scalability**
  - Current procedures inflexible
    - Need lots of data in description array
  - Possible solution to develop predefined AS templates
    - Just specify AS “type” and subnet



```
set opt(PID) "root-emu1"
set opt(EID) "in1"
set opt(NODE_TYPES) {router server host}
set opt(ROUTER_HARDWARE) pcvm
set opt(SERVER_HARDWARE) pcvm
set opt(HOST_HARDWARE) pcvm
set opt(NUM_ROUTERS) 60
set opt(NUM_SERVERS) 3
set opt(NUM_HOSTS) 18

# Need to use {} to disable early parsing of STARTCMD
set opt(NODE_STARTCMD)
    {/proj/$opt(PID)/exp/$opt(EID)/config/configure.pl --debug >& \
    /proj/$opt(PID)/exp/$opt(EID)/config/log/$node_type-$node_number.log}
set opt(NODE_TARFILES) ""

set opt(CONFIG_DEST_BASE) "/opt/etc"

# networks
set opt(NETWORKS) "network_64601 network_64602 network_64603"
```



```
# 223.0.0.0/20 : 64601
# 223.0.15.0/24 infrastructure
# 223.0.15.223/27 loopback
# 223.0.15.160/27 core point-to-point
# 223.0.15.128/27 border point-to-point
# 223.0.15.0/25 naps
set opt(network_64601_ROUTING_CORE_ROUTERS) "01 05 09 13 17"
# 223.0.15.0/29 nap01
set opt(network_64601_ROUTING_01_LOOPBACK) "223.0.15.254"
set opt(network_64601_ROUTING_01_IGP) "1:223.0.15.0/29 0:223.0.15.128/30 0:223.0.15.160/30
0:223.0.15.164/30 0:223.0.15.168/30 0:223.0.15.184/30"
set opt(network_64601_ROUTING_01_BORDER_ROUTERS) "02"
set opt(network_64601_ROUTING_01_RR_CLIENTS) "03 04"
set opt(network_64601_ROUTING_01_OTHER_ROUTERS) "05 09 13 17"
set opt(network_64601_ROUTING_01_02_LOOPBACK) "223.0.15.253"
set opt(network_64601_ROUTING_01_02_IGP) "0:223.0.15.128/30"
set opt(network_64601_ROUTING_01_02_PEERS) ""
set opt(network_64601_ROUTING_01_03_LOOPBACK) "223.0.15.252"
set opt(network_64601_ROUTING_01_03_IGP) "1:223.0.15.0/29"
set opt(network_64601_ROUTING_01_03_CUSTOMERS) "21"
set opt(network_64601_ROUTING_01_03_21_ASN) "64701"
set opt(network_64601_ROUTING_01_03_21_RID) "10.254.0.2"
set opt(network_64601_ROUTING_01_03_21_PEER) "10.254.0.1"
set opt(network_64601_ROUTING_01_03_21_HOSTS) "01 02"
set opt(network_64601_ROUTING_01_03_21_SERVERS) ""
set opt(network_64601_ROUTING_01_03_21_NETWORKS) "223.0.0.0/29"
set opt(network_64601_ROUTING_01_04_LOOPBACK) "223.0.15.251"
set opt(network_64601_ROUTING_01_04_IGP) "1:223.0.15.0/29"
set opt(network_64601_ROUTING_01_04_CUSTOMERS) ""
```



```
# 223.0.16.0/20 : 64602
# 223.0.31.0/24 infrastructure
# 223.0.31.223/27 loopback
# 223.0.31.160/27 core point-to-point
# 223.0.31.128/27 border point-to-point
# 223.0.31.0/25 naps
set opt(network_64602_ROUTING_CORE_ROUTERS) "26 30 34 38 42"
# 223.0.31.0/29 nap26
set opt(network_64602_ROUTING_26_LOOPBACK) "223.0.31.254"
set opt(network_64602_ROUTING_26_IGP) "1:223.0.31.0/29 0:223.0.31.128/30 0:223.0.31.160/30
0:223.0.31.164/30 0:223.0.31.168/30 0:223.0.31.184/30"
set opt(network_64602_ROUTING_26_BORDER_ROUTERS) "27"
set opt(network_64602_ROUTING_26_RR_CLIENTS) "28 29"
set opt(network_64602_ROUTING_26_OTHER_ROUTERS) "30 34 38 42"
set opt(network_64602_ROUTING_26_27_LOOPBACK) "223.0.31.253"
set opt(network_64602_ROUTING_26_27_IGP) "0:223.0.31.128/30"
set opt(network_64602_ROUTING_26_27_PEERS) ""
set opt(network_64602_ROUTING_26_28_LOOPBACK) "223.0.31.252"
set opt(network_64602_ROUTING_26_28_IGP) "1:223.0.31.0/29"
set opt(network_64602_ROUTING_26_28_CUSTOMERS) ""
set opt(network_64602_ROUTING_26_29_LOOPBACK) "223.0.31.251"
set opt(network_64602_ROUTING_26_29_IGP) "1:223.0.31.0/29"
set opt(network_64602_ROUTING_26_29_CUSTOMERS) "46"
set opt(network_64602_ROUTING_26_29_46_ASN) "64801"
set opt(network_64602_ROUTING_26_29_46_RID) "10.254.1.2"
set opt(network_64602_ROUTING_26_29_46_PEER) "10.254.1.1"
set opt(network_64602_ROUTING_26_29_46_HOSTS) "09 10"
set opt(network_64602_ROUTING_26_29_46_SERVERS) ""
set opt(network_64602_ROUTING_26_29_46_NETWORKS) "223.0.16.0/29"
```



```
proc build_nodes {ns_ref opt_ref} {
    # FIXME: hardcoded to expect $router, $host, and $server arrays.
    # This needs to somehow be flexible around NODE_TYPES.
    global router host server
    upvar $ns_ref ns
    upvar $opt_ref opt

    # NODE_TYPES contains a list of types.
    foreach my_node_type $opt(NODE_TYPES) {
        set ${my_node_type}(0) 1
        # NUM_${my_node_type}S contains the number of nodes for a node type.
        set num_node_type NUM_
        append num_node_type [string toupper $my_node_type] S
        for {set i 1} {$i <= $opt($num_node_type)} {incr i} {
            set this_node_number [format "%02d" $i]
            set ${my_node_type}($this_node_number) [$ns node]
            set this_node_hardware [string toupper $my_node_type]
            append this_node_hardware _HARDWARE
            tb-set-hardware [set ${my_node_type}($this_node_number)] $opt($this_node_hardware)
            ## FIXME: startcmd and tarfiles are not working.
            ## I believe the cause is that its trying to reference a node
            ## that the parse is unable to locate.
            #tb-set-node-startcmd [set ${my_node_type}($this_node_number)] $opt(NODE_STARTCMD)
            #tb-set-node-tarfiles [set ${my_node_type}($this_node_number)] $opt(NODE_TARFILES)
        }
    }
}
```



```
proc build_nap_1 {ns_ref opt_ref network_id core_router} {
  # FIXME: hardcoded to expect $router, $host, and $server arrays.
  # This needs to somehow be flexible around NODE_TYPES.
  global juniper router host server
  upvar $ns_ref ns
  upvar $opt_ref opt
  append opt_idx $network_id _ROUTING_ $core_router

  # Build the nap lan connecting the core router and route reflector clients.
  set lan_routers [list $router($core_router)]
  # ${core_router}_RR_CLIENTS contains the list of
  # route reflector clients for the core router.
  foreach rr_client $opt(${opt_idx}_RR_CLIENTS) {
    # ${rr_client}_CUSTOMERS contains a list of customers for the rr client.
    foreach customer $opt(${opt_idx}_${rr_client}_CUSTOMERS) {
      build_customer ns opt ${opt_idx}_${rr_client} $rr_client $customer
    }

    lappend lan_routers $router($rr_client)
  }
  # The first item of the IGP list contains the network of the nap lan.
  set igp_idx 0
  # Strip off the prepended AREA: to get just the network and prefix.
  regexp {[0-9]+:(.*)} [lindex $opt(${opt_idx}_IGP) $igp_idx] match lan_network

  build_lan ns ${network_id}_nap${core_router} $lan_routers $lan_network

  # Build the links between the nap core router and any
  # border routers in the nap or other core routers on the network.
  append link_id $network_id _nap $core_router _ $core_router
}
```



```
# ${core_router}_BORDER_ROUTERS contains a list of
# border routers in the nap.
foreach border_router $opt(${opt_idx}_BORDER_ROUTERS) {
  # ${border_router}_PEERS contains a list of peers for the border router.
  foreach peer $opt(${opt_idx}_${border_router}_PEERS) {
    build_border ns opt ${opt_idx}_${border_router} $border_router $peer
  }
  # All border router links come after the nap network and before
  # any other core router links in the IGP list.
  incr igp_idx
  # Strip of the prepended AREA: to get just the network and prefix.
  regexp {[0-9]+:(.*)} [lindex $opt(${opt_idx}_IGP) $igp_idx] \
    match router_network
  build_link ns $link_id$border_router \
    $router($core_router) $router($border_router) $router_network
}
```

```
# ${core_router}_OTHER_ROUTERS contains a list of
# other core routers in the network.
foreach other_router $opt(${opt_idx}_OTHER_ROUTERS) {
  # All other core router links come after the
  # border router links in the IGP list.
  incr igp_idx
  # Strip of the prepended AREA: to get just the network and prefix.
  regexp {[0-9]+:(.*)} [lindex $opt(${opt_idx}_IGP) $igp_idx] \
    match router_network
  build_link ns $link_id$other_router \
    $router($core_router) $router($other_router) $router_network
}
}
```



