Design of the New and Improved NetSpec Controller

Radhakrishnan R. Mukkai

Masters Thesis Defense

Date: 8 Jan 2004

Committee:

Dr. Jerry James (Chair)

Dr. Douglas Niehaus

Dr. David Andrews

Information and Telecommunication Technology Center

University of Kansas

Presentation Outline

- Introduction
- Why a new NetSpec controller?
- Goals
- Background Information
- NetSpec 6.0
- Implementation
- Testing
- Related Work
- Conclusions and Future Work

Introduction

What is NetSpec?

- Software tool designed by researchers at KU.
- Provides a framework to centrally control daemons running on other machines.
- Provides a simple, block structured language for specifying experimental parameters.

Features of NetSpec

- Centralized control and command system
- Scalability
- Reproducibility
- Extensibility

Why a new NetSpec controller

The older NetSpec controller (version 5.0) was

- Complex
- Non-modular

Reasons

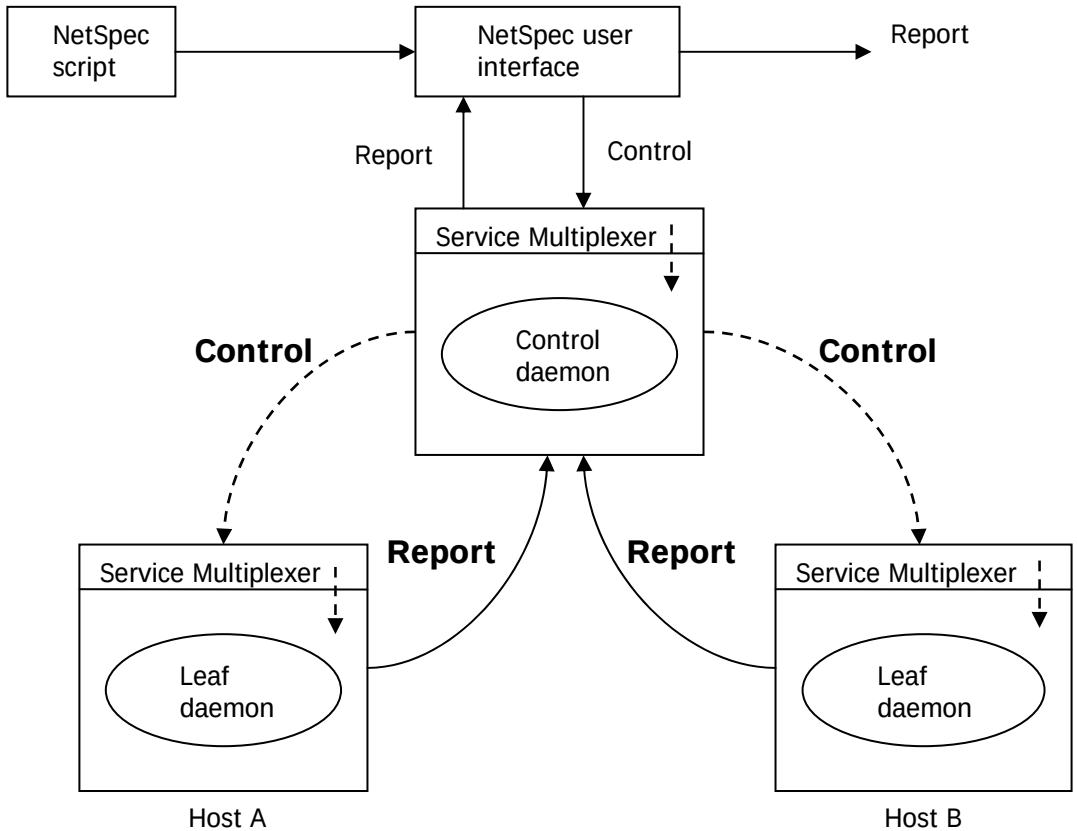
- Parsing & control operations performed simultaneously.
- No data structures in the controller to store parsed values.

NetSpec sample script

```
netTraffic testbed61 {
  type = full (blocksize=32768,
   duration=10);
  protocol = tcp;     # daemon data
        ...
}
```

- Controller looks at the daemon data as *parameter = value* pairs

NetSpec 5.0 Architecture



Goals – NetSpec 6.0

- NetSpec user interface not needed. User script should be given as input to the controller.
- Controller now responsible for performing the various control operations previously initiated by the NetSpec user interface.
- Separate parsing and control operations. Data structures are used as the “glue” to tie them together.

Background Information

Daemons accomplish their tasks in phases.

Two mechanisms to schedule the phases of daemons:

- Classic 8 Phase
- Variable Phase

Daemons and controller communicate using a text-based protocol, called *Remote Control and Information Protocol (RCIP)*

Background Information

Classic 8 Phase Scheduling Mechanism

- All the daemons are executed in 8 phases.
- Controller sends a text command to the daemon, thus initiating a phase.
- Daemons reply with an Acknowledgement indicating completion of the phase.

NetSpec Classic 8 Phase

Command	Action
Setup	Allocate Resources
Open	Establish necessary socket connections
Run	Execute the desired function
Close	Close all the socket connections
Finish	Finish the execution
Report	Prepare and send the report
Teardown	Release all the acquired resources
Kill	Terminate the execution

Background Information

Daemon functions can be executed either in serial or parallel fashion using the following execution constructs – *Serial, Parallel or Cluster.*

Example script:

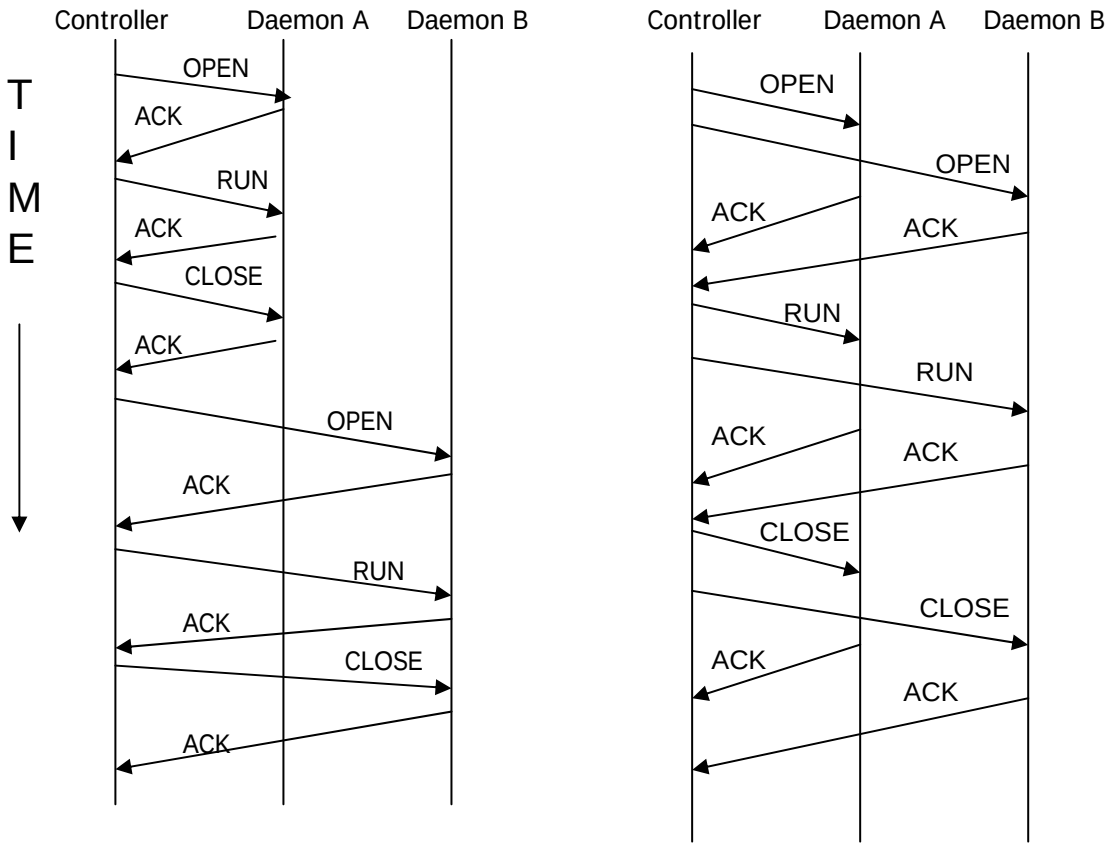
```
serial {   # Can be either serial, parallel or cluster
      netTraffic testbed61 {
        # daemon data
       }
      netTraffic testbed61 {
        # daemon data
       }
}
```

Execution Construct

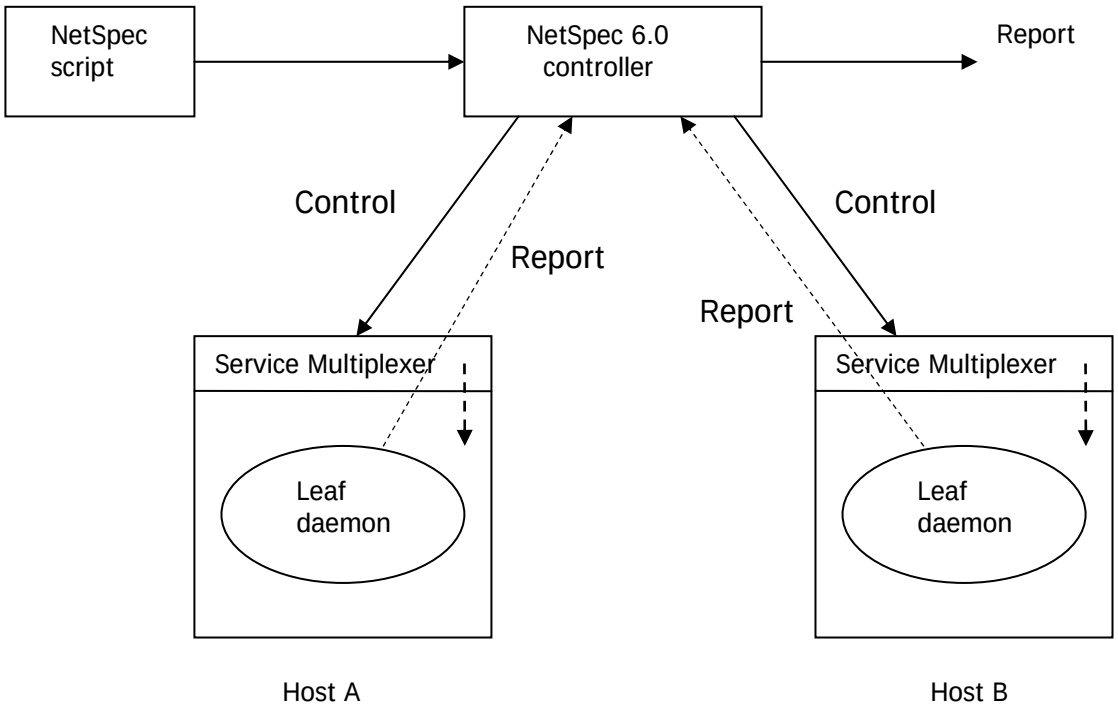
Let us consider two daemons A and B

- Serial: The *open, run* and *close* phases with respect to a daemon (say, daemon A) are finished before daemon B is executed.
- Parallel: The *run* phases of daemons A and B are executed concurrently.
- Cluster: The *open, run* and *close* phases are executed concurrently.

Serial/Cluster Construct

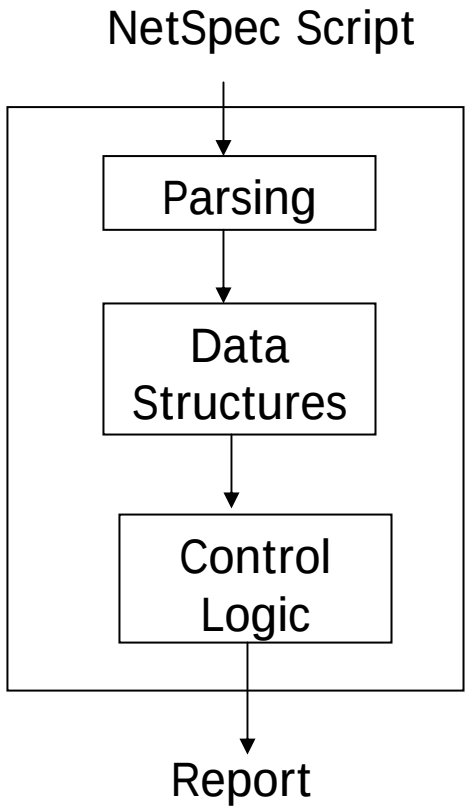


NetSpec 6.0 Architecture



Implementation

NetSpec Controller Design



Implementation

Parsing

- Lex/YACC used to parse the NetSpec script.

Control Logic

- Connect to the peer (TCP sockets).
- Invoke the daemon.
- Pass parameter-value pairs.
- Executing the phases.

Features of NetSpec 6.0 Architecture

Simpler

- NetSpec user interface has been removed.
- NetSpec 5.0 supported a multi-level control hierarchy. Majority of the experiments utilized the single-level control hierarchy.

Modular

- Parsing and control functionalities are not tied together.
- Data structures used to store parsed values.

NetSpec Variable Phase

Addresses the following shortcoming with the Classic 8 phase mechanism:

- Daemons were limited to eight phases.
- All daemons had to go through their execution in eight phases. This resulted in daemons doing nothing in certain phases.
- Only the *open, run* and *close* phases were subject to various execution modes.

NetSpec Variable Phase

Variable Phase Features

- NetSpec user can choose the number of execution phases for each daemon.
- Slot-based mechanism for scheduling the phases of various daemons.

Slot-based Mechanism

- NetSpec experiment is divided into various slots.
- Each slot contains phases of daemons which are to be executed concurrently.
- Slots themselves are executed serially.

NetSpec 5.0 Variable Phase Features

- Execution constructs controlled how phases of daemons in a slot would be executed.
- Schedule information was passed in a separate file.
- Two different parsers were used to parse the schedule and daemon information.

NetSpec 6.0 Variable Phase Features

- Two-level schedule information specifies - how individual daemons and phases of daemons will be scheduled.
- Execution constructs *serial* and *parallel* are supported.
- Schedule file made part of the NetSpec script.
- Single parser for both schedule and daemon information

NetSpec 6.0 Sample Script

```
#SCHEDULING INFORMATION
map defmap {
  daemon = new_daemon (daemon_name = template;
          phaseA = 1;
          phaseB = 2;
          phaseC = 3;);
}
#DAEMON INFORMATION
serial {
   template testbed13 {
       # parameter-value pairs
   }

  template testbed14 {
      # parameter-value pairs
  }
}
```

NetSpec Variable Phase

- NetSpec controller parses the schedule information and constructs command strings.
- These command strings are passed to the daemon when a particular phase needs to be initiated.
- Command strings have the format:

  Daemon-name:phase to initiate

  Ex: *template:phaseA*

Scheduling Daemons

```
map defmap {
  daemon = new_daemon (daemon_name = daemonA;
            setupCommand=1;
            runCommand=3;
            finishCommand=4;);  }

map defmap {
  daemon = new_daemon (daemon_name = daemonB;
            setupCommand=1;
            runCommand=3;
            finishCommand=5;);  }

map defmap {
  daemon = new_daemon (daemon_name = daemonC;
            setupCommand=2;
            openCommand=3;
            runCommand=4;
            finishCommand=6;); }
```

Scheduling Daemons

SLOT 1: daemonA:setupCommand and
daemonB:setupCommand
SLOT 2: daemonC:setupCommand
SLOT 3: daemonA:runCommand,
daemonB:runCommand and
daemonC:openCommand
SLOT 4: daemonA:finishCommand and
daemonC:runCommand
SLOT 5: daemonB:finishCommand
SLOT 6: daemonC:finishCommand

NetSpec Script - Daemon Information

```
serial {    # can be either serial or parallel
  daemonA testbed1 {
    # specify parameter value pairs associated with daemonA
  }
  daemonA testbed2 {
    # specify parameter value pairs associated with daemonA
  }
  daemonB testbed3 {
     # specify parameter value pairs associated with daemonB
  }
  daemonB testbed4 {
     # specify parameter value pairs associated with daemonB
  }
  daemonC testbed5 {
     # specify parameter value pairs associated with daemonC
  }
  daemonC testbed6 {
     # specify parameter value pairs associated with daemonC
  }
}
```

Serial Construct

daemonA on testbed1	daemonA on testbed2	daemonB on testbed3	daemonB on testbed4	daemonC on testbed5	daemonC on testbed6
setup					
run					
finish					
	setup				
	run				
	finish				
		setup			
		run			
		finish			

TIME

Parallel Construct

TIME

daemonA on testbed1	daemonA on testbed2	daemonB on testbed3	daemonB on testbed4	daemonC on testbed5	daemonC on testbed6
setup	setup	setup	setup		
				setup	setup
run	run	run	run	open	open
finish	finish			run	run
		finish	finish		
				finish	finish

New NetSpec Script Language Features

Schedule Information

- NetSpec script becomes the one and only source for all information.

Transferring Files to/from Daemons

- Capabilities were added to send/receive multiple files to/from the daemons.
- Each daemon uses file as input for its processing tasks.
- Daemon can send back results of processing through file(s) to controller.
- Both text/binary files can be handled.

Transfer Files to/from daemons

- Daemon options section added to specify files to be sent and received.

Example script file:

```
parallel {
    nssyscmd NodeB (to_file="echo.txt":"/tmp/echo.txt",
     from_file="test.txt":"/tmp/test.txt") {
                    # parameter-value pairs
                    # ....
   }
```

- *to_file* parameter specifies files to be transferred to the daemon.
- *from_file* parameter specifies files to be transferred from the daemon.

New NetSpec Script Language Features

Daemon/host identifier

- Provision for uniquely identifying a daemon/host pair has been provided.

Example script:

```
netTraffic testbed61 daemon_instance1 {
   ...
}

netTraffic testbed61 daemon_instance2 {
   ...
}
```

Comments

- Comments start with a # and continue to the end of a line

Testing

Testing Correctness

- Parsing – *Pretty Printer option*
- Control Logic

Testing Robustness

- Tested framework by executing scripts involving 40 daemons on 8 hosts.
- Report data generated was around 18 MB.

Related Work

- TTCP
- Iperf
- Netperf

Conclusions

Three goals that we set out to achieve were reached:

- Simplicity
- Modularity
- Robustness

Capabilities for downloading files to/from the daemon were provided.

Options like *Pretty Printer* provided to test controller code better.

Future Work

- Semantic *Group* which represents schedule and daemon information.
- Script could have various *Groups* and information about scheduling the various *Groups.*

Questions or Comments

?? or !!

Thank you!!