# Scripting the Linux Routing Table with Lua

Lourival Vieira Neto, Marcel Moura,
Ana Lúcia de Moura and Roberto Ierusalimschy

Ring-0 Networks and Departamento de Informática, PUC-Rio
Netdev 0x17

# Agenda

ring-∅

- Introduction
- Original Architecture
- Following Implementation
- Modular Implementation
- Adaptive Routing Service
- Final Remarks

# Introduction

Lunatik is a framework for scripting the Linux kernel with the Lua programming language (since 2008)

- Use Cases
  - CPUfreq
  - NFLua
    - Used by network operators
    - Advanced cybersecurity and network monitoring
    - 20 million home routers
  - XDPLua
    - Running in PoPs with 10 Gbps bandwidth
    - Peaks of around 5.3 Gbps , only up to 4% of CPU
  - ULP-Lua
  - kTLS-Lua
- Related Work
  - NetBSD's lua(4)
    - NPFLua
    - DDB
    - Kauth
  - ZFS
  - PacketScript
  - Memcached
  - pNFS

# Introduction

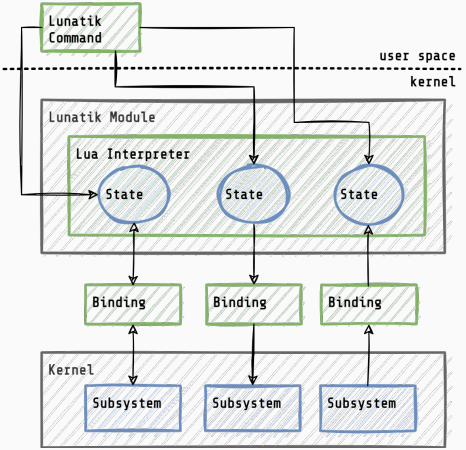
## Why Lua?

- Small (250 KB) and fast scripting language
- Widely used for scripting network monitoring and security tools
  - e.g., Wireshark, Nmap, Snort
- Sandboxing techniques (Netdev 0x14)

## But... really? Why Lua?

- It's easy !
  - No toolchain dependencies
  - Very simplified workflow

# Original Architecture

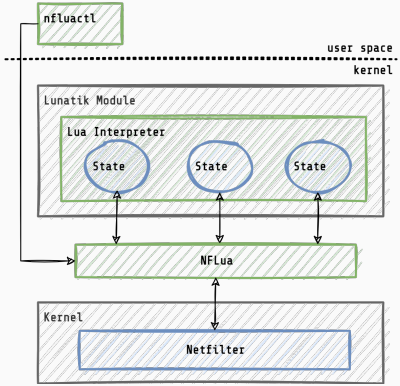



- Suitable only for process context
  - e.g., CPUfreq

# Original Architecture (CPUfreq)

```
up          = 80
down        = 30
overheated = 100

function throttle(cpu, cur, max, min)
  -- get utilization since last check
  local load = get_load(cpu)

  -- get temperature
  local temp = acpi.get_temp(cpu)

  if temp >= overheated then
    -- decrease frequency by 20%
    cpufreq.target(cpu, cur * 80 / 100, '<=')
  else
    if load > up then
      -- rise frequency to the maximum value
      cpufreq.target(cpu, max, '>=')

    elseif load < down then
      -- decrease frequency by 20%
      cpufreq.target(cpu, cur * 80 / 100, '<=')
    end
  end
end
```

# Following Implementation



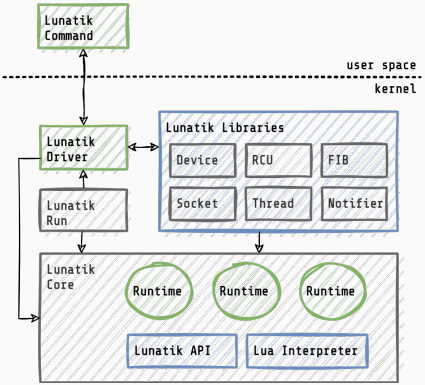

- Ad hoc scripting environment
  - Software interrupt context
  - e.g., Netfilter , XDP

# Previous Implementation (Netfilter/XDP)

```
function checkuseragent(pkt)
  -- extracts User-Agent HTTP header
  local pattern = "User%-Agent:%s(.-)\r\n"
  local useragent = string.match(pkt, pattern)

  return blocklist[useragent]
end
```

```
function checkcookie(pkt, saddr)
  -- checks if challenge is not set yet (first request)
  if not cookies[saddr] then
    return true
  end

  -- extracts __xdp cookie
  local pattern = "Cookie:%s*=__xdp=(%d+)%s*"
  local cookieval = string.match(tostring(pkt), pattern)

  -- checks cookie's value
  return cookies[saddr] == cookieval
end
```

# Modular Implementation




- Richer scripting environment
  - Base environment for the kernel as whole
  - Process and software interrupt context
  - Partially developed in Lua
  - Loads files synchronously from the FS
  - New set of libraries

# Modular Implementation (Konami Code)

ring-0

```
local notifier = require("notifier")

-- <UP> <UP> <DOWN> <DOWN> <LEFT> <RIGHT> <LEFT> <RIGHT> <LCTRL> <LALT>
local konami = {code = {103, 103, 108, 108, 105, 106, 105, 106, 29, 56}, ix = 1}
function konami:completion(key)
  self.ix = key == self.code[self.ix] and (self.ix + 1) or 1
  return self.ix == (#self.code + 1)
end

local notify = notifier.notify
local kbd    = notifier.kbd
local enable = true
local function locker(event, down, shift, key)
  if not down and event == kbd.KEYCODE and konami:completion(key) then
    enable = not enable
  end
  return enable and notify.OK or notify.STOP
end

notifier.keyboard(locker)
```

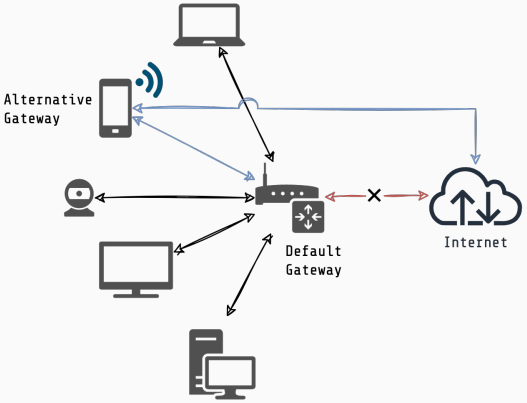
# Modular Implementation (Konami Code-Fu)

ring-0

```
local notifier = require("notifier")
local device   = require("device")
local rcu      = require("rcu")

local log = rcu.table()
rcu.publish("konamifu.log", log)

local function nop() end
local konamifu = {name = "konamifu", open = nop, release = nop}

function konamifu:read()
  return string.format("my foo is NOT that better (%s)\n", log.cnt)
end

-- <UP> <UP> <DOWN> <DOWN> <LEFT> <RIGHT> <LEFT> <RIGHT> <LCTRL> <LALT>
local konami = {code = {103, 103, 108, 108, 105, 106, 105, 106, 29, 56}, ix = 1}
function konami:completion(key)
  self.ix = key == self.code[self.ix] and (self.ix + 1) or 1
  return self.ix == (#self.code + 1)
end

local notify = notifier.notify
local kbd    = notifier.kbd
log.cnt      = 0
local function locker(event, down, shift, key)
  if not down and event == kbd.KEYCODE and konami:completion(key) then
    log.cnt = log.cnt + 1
  end
end

notifier.keyboard(locker)
device.new(konamifu)
```

```
$ lunatik
Lunatik 3.2  Copyright (C) 2023 ring-0 Ltda.
> rcu = require("rcu")
> log = rcu.subscribe("konamifu.log")
> return log.cnt
1
^D
$ cat /dev/konamifu
my foo is NOT that better (1)
```

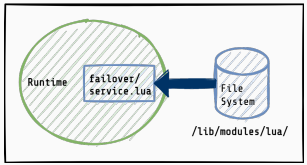
# Adaptive Routing Service




- Heartbeat latency
  - UDP
- Enable/disable alternative GW
  - TCP

# Adaptive Routing Service

```
$ lunatik run failover/service
```




```
local script = "failover/service"
lunatik.__runtimes[script] = lunatik.runtime(script)
```

# Adaptive Routing Service (Notification Handler)

```
local lunatik  = require("lunatik")
local thread   = require("thread")
local notifier = require("notifier")
local inet     = require("socket.inet")
local fib      = require("fib")
local rcu      = require("rcu")
local conf     = require("failover.conf")

local netdev = notifier.netdev
local notify = notifier.notify

local elected = rcu.table(8)
rcu.publish("elected", elected)

local function alt_gateway(state)
  local client <close> = inet.tcp()
  client:connect(elected.ip, conf.port)
  client:send(state)
end
```

```
local state = "down"
local function failover(event, ifname)
  if ifname ~= conf.wan then return notify.OK end

  if event == netdev.UP then
    state = "up"
  elseif event == netdev.CHANGE then
    if state == "up" then
      state = "down"
      fib.newrule(conf.table_id, conf.priority)
      alt_gateway("up")
    else
      state = "up"
      fib.delrule(conf.table_id, conf.priority)
      alt_gateway("down")
    end
  end
end

notifier.netdevice(failover)

local daemon = "failover/daemon"
thread.run(lunatik.runtime(daemon), daemon)
```

# Final Remarks

ring-0

- Lua extensions directly on top of the framework
  - Instead of calling scripts from bindings for kernel subsystems
  - Easier workflow
- New facilities for creating libraries
  - Allows extending the framework itself
- New kernel service with only ~80 LOC
- Interoperates with other Lua kernel-scripting environments
  - NFLua
  - XDPLua
  - ZFS

github.com/luainkernel/lunatik

README.md

# Lunatik

Lunatik is a framework for scripting the Linux kernel with Lua. It is composed by the Lua interpreter modified to run in the kernel; a device driver (written in Lua =)) and a command line tool to load and run scripts and manage runtime environments from the user space; a C API to load and run scripts and manage runtime environments from the kernel; and Lua APIs for binding kernel facilities to Lua scripts.

Here is an example of a character device driver written in Lua using Lunatik to generate random ASCII printable characters:

```
-- /lib/modules/lua/passwd.lua
--
-- implements /dev/passwd for generate passwords
-- usage: $ sudo lunatik run passwd
--        $ head -c <width> /dev/passwd

local device = require("device")
local linux  = require("linux")

local function nop() end -- do nothing

local s = linux.stat
local driver = {name = "passwd", open = nop, release = nop, mode = s.IRUGO}

function driver:read() -- read(2) callback
        -- generate random ASCII printable characters
        return string.char(linux.random(32, 126))
end

-- creates a new character device
device.new(driver)
```