Let's Build a BGP Traffic Engineering Controller

NANOG 71 Hackathon Tutorial
30 September 2017

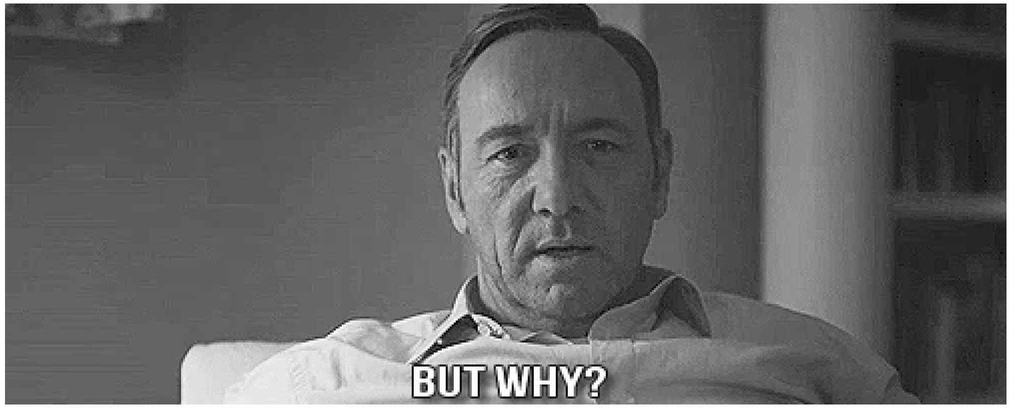
Brandon Bennett
Network Robot Mechanic, Facebook

What are we building

We are going to build the framework for a BGP-based Traffic Engineering controller

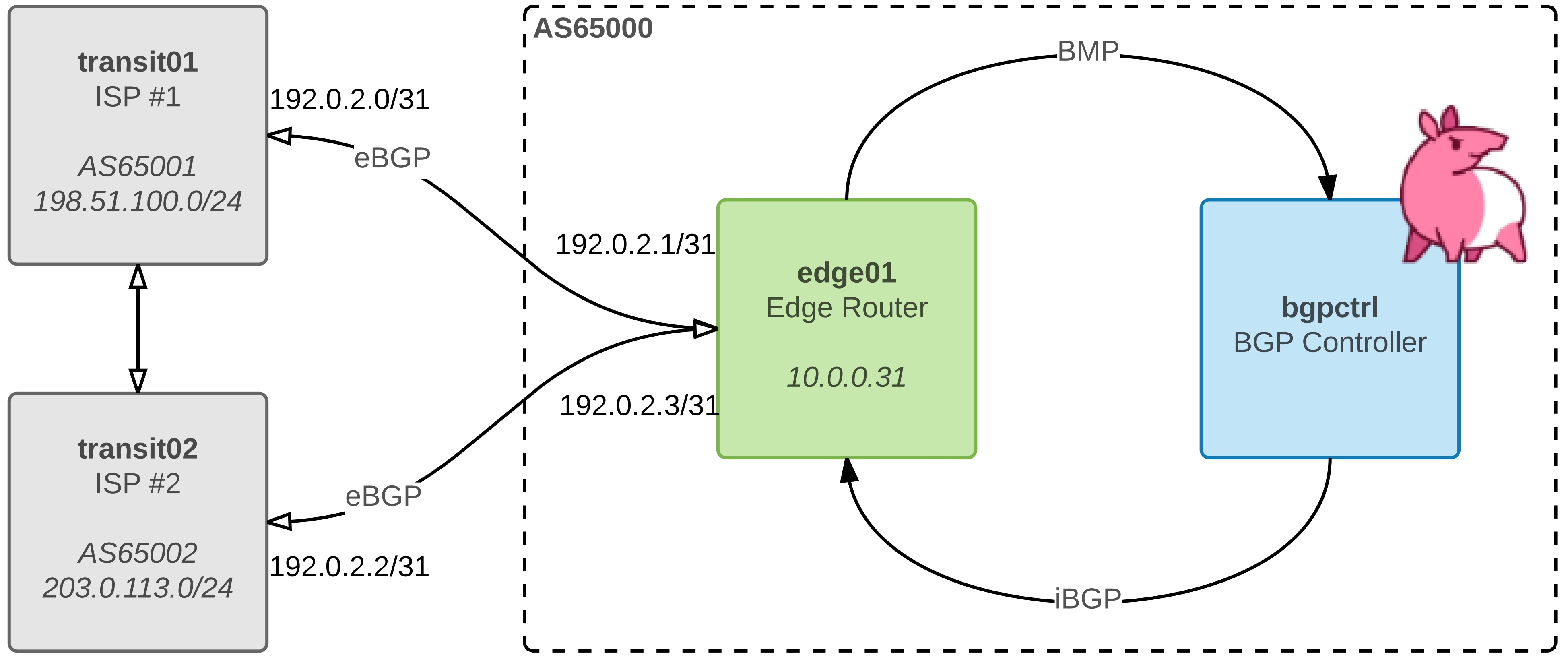
- Push routes over iBGP from our controller to a peer router
- Read peer router neighbor state and routes from BGP Monitoring Protocol (BMP)
- Scale this to many routers!

But why?



- Dynamically monitor BGP neighbors
- Control our egress traffic dynamically based on packet loss, latency, cost, application, etc outside of standard static BGP policy

Diagram


transit01
ISP #1
AS65001
198.51.100.0/24
192.0.2.0/31
eBGP
transit02
ISP #2
AS65002
203.0.113.0/24
eBGP
192.0.2.2/31
AS65000
192.0.2.1/31
192.0.2.3/31
edge01
Edge Router
10.0.0.31
BMP
bgpctrl
BGP Controller
iBGP

The tools

- Go programming language - golang.org (http://golang.org)
- GoBGP library/daemon - osrg.github.io/gobgp/ (https://osrg.github.io/gobgp/)
- network.toCode() virtual labs - labs.networktocode.com (http://labs.networktocode.com)

My Editor

- Visual Studio Code code.visualstudio.com/ (https://code.visualstudio.com/) with the Go plugin
- Go vscode extension marketplace.visualstudio.com/items?itemName=lukehoban.Go (https://marketplace.visualstudio.com/items?itemName=lukehoban.Go)

Why Go?

We are going to use the Go programming language to build this controller.

- Compiled
- Statically typed
- Easy to program with good stdlib and third party support
- Great and easy concurrency built into the runtime (perfect for networks and daemons!)
- Fun

Although you could write this in other languages too

Learning Go

Installing Go

Windows:

- Download and run MSI installer: golang.org/dl/ (https://golang.org/dl/)

OS X:

- Download the offical package installer
- Install from homebrew

```
$ brew install golang
```

Linux:

- Download binaries from https//golang.org/dl and put them somewhere:(i.e $HOME/go1.9 or /opt/go)
- Use your distro's package (apt, dnf, etc)

See golang.org/doc/install (https://golang.org/doc/install) for more help

Creating our GOPATH

GOPATH is our workspace for coding in and downloading dependencies to. In Go 1.9 the default is $HOME/go

```
export GOPATH=$HOME/go
```

There can be up to three subfolders under this GOPATH

```
$GOPATH
  ├── src     # Where your code goes
  ├── bin     # Where go will store some binaries
  └── pkg     # Where go stores compiled code
```

Code we write should go under `src` (dependencies will be dl here too!)

My first Go program.

```go
package main

import "fmt"

func main() {
    who := "NANOG 71"
    fmt.Printf("Welcome %s\n", who)
}
```

Run

Functions

```
func add(a int, b int) int {
    return a + b
}

func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    a, b := swap("Juniper", "Cisco")
    x := add(40, 2)
    fmt.Println(a, b, x)
}
```

Run

Variables

```
func main() {
    var a int // Must define type
    b := 30   // type infered (is int)

    var x string
    y := "MPLS Rulez"

    fmt.Printf("a: %d  \nb: %d\n", a, b)
    fmt.Printf("x: %q  \ny: %q\n", x, y)
    fmt.Printf("Type of b: %T\n", b)
    fmt.Printf("Type of y: %T\n", y)
}
```

Run

Global variables and constants

```go
package main

import "fmt"

const MaxLatency = 0.15

var FavoriteProtcol = "BGP"

const (
    Cisco = iota
    Juniper
    Arista
)

func main() {
    fmt.Printf("Max Latency: %f,  Favorite Protocol: %s\n", MaxLatency, FavoriteProtcol)
    fmt.Println(Cisco, Juniper, Arista)
}
```

Run

Slices and Maps

We can also store collections of data

```go
func main() {
    ports := []int{22, 179, 80, 443}
    fmt.Println(ports[2])
    fmt.Println(ports[1:])

    mapNames := map[int]string{
        22:  "ssh",
        179: "bgp",
    }
    fmt.Println(mapNames[22])
    fmt.Println(mapNames[179])
}
```

Run

Looping

There is only one keyword for looping in Go and that is `for`

```
func main() {
    // C Style
    for i := 0; i < 10; i++ {
        fmt.Println(i)
    }

    // Loop over slice or map
    protocols := []string{"BGP", "OSPF", "RIP"}
    for _, p := range protocols {
        fmt.Println(p)
    }

    // Loop forever!
    for {
        fmt.Println("LEEEEETTTS GO!")
        time.Sleep(1 * time.Second)
    }
}
```

Run

Creating Object

We can create custom objects as well using a `struct`

```go
package main

import "fmt"

type Router struct {
    Hostname  string
    IPAddress string
    ASN       int
}

func main() {
    r := Router{
        Hostname:  "edge01.sjc1",
        IPAddress: "2001:db8::/32",
    }
    r.ASN = 32934

    fmt.Printf("%s:\n\tIP: %s\n\tASN: %d", r.Hostname, r.IPAddress, r.ASN)
}
```

Run

Methods

We can also attach methods to a any type

```
func (r *Router) String() string {
    return fmt.Sprintf("%s (AS%d) - %s", r.Hostname, r.ASN, r.IPAddress)
}

func (r *Router) ISPrivateASN() bool {
    return r.ASN >= 65000 && r.ASN <= 65535
}

func main() {
    r := Router{
        Hostname:  "edge01.sjc1",
        IPAddress: "2001:db8::/32",
        ASN:       65000,
    }
    fmt.Println(r.String())
    fmt.Println(r.ISPrivateASN())
}
```

Run

Go goroutines

Go has the ability to run any function in the "background". This is analogous to the & in Unix shells like bash

```
$ ./some_command      # Run in foreground
$ ./some_command &    # Run in background
```

In go we use the go keyword before the function

```
func dostuff() {
    fmt.Println("Start dostuff()")
    time.Sleep(1 * time.Second)
    fmt.Println("End dostuff()")
}

func main() {
    go dostuff() // Don't block, run in background
    fmt.Println("From main()")
    time.Sleep(2 * time.Second)
}
```

Run

Channels

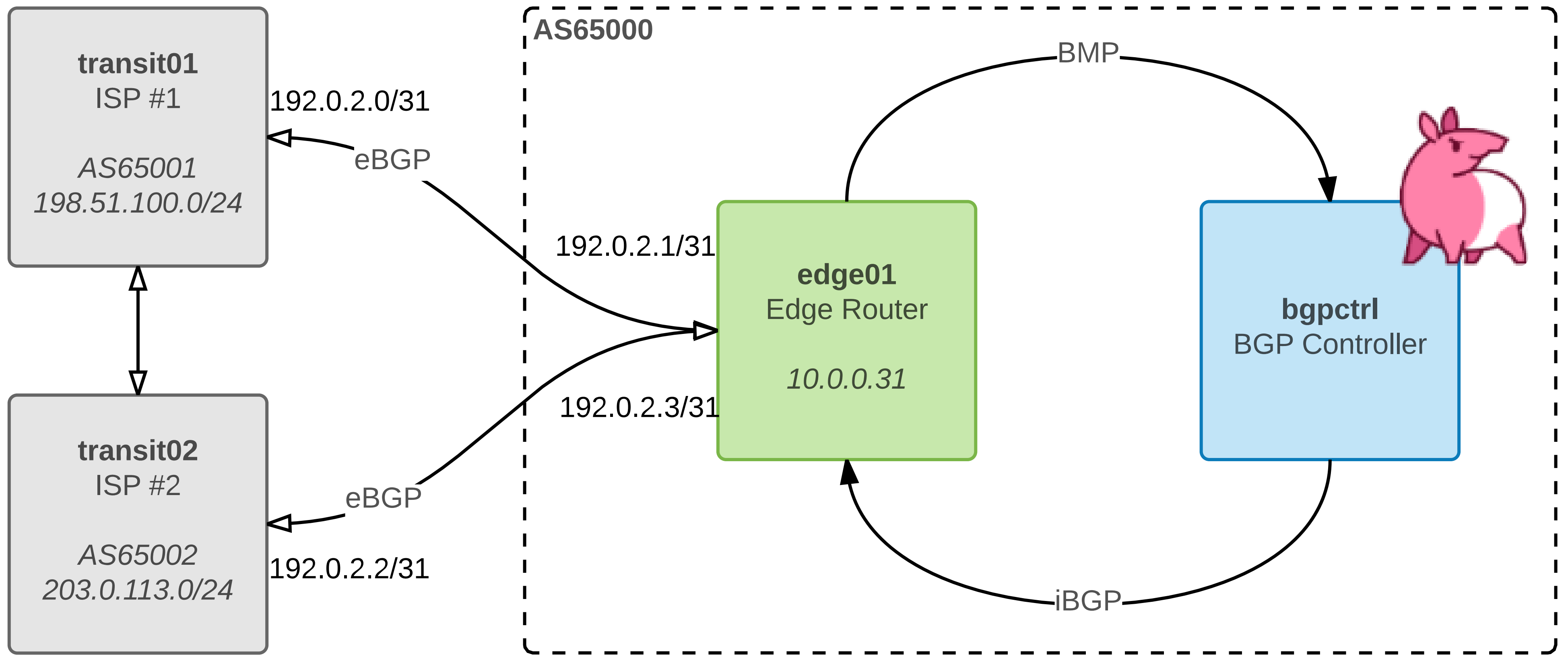
Channels are used to send data between and synchronize goroutines*

```
func main() {
    data := make(chan int)

    go func() {
        time.Sleep(2 * time.Second)
        data <- 1
        fmt.Println("Done sending data")
    }()

    fmt.Println("Waiting for data")
    val := <-data
    fmt.Printf("got: %d\n", val)
}
```

Run

Our environment

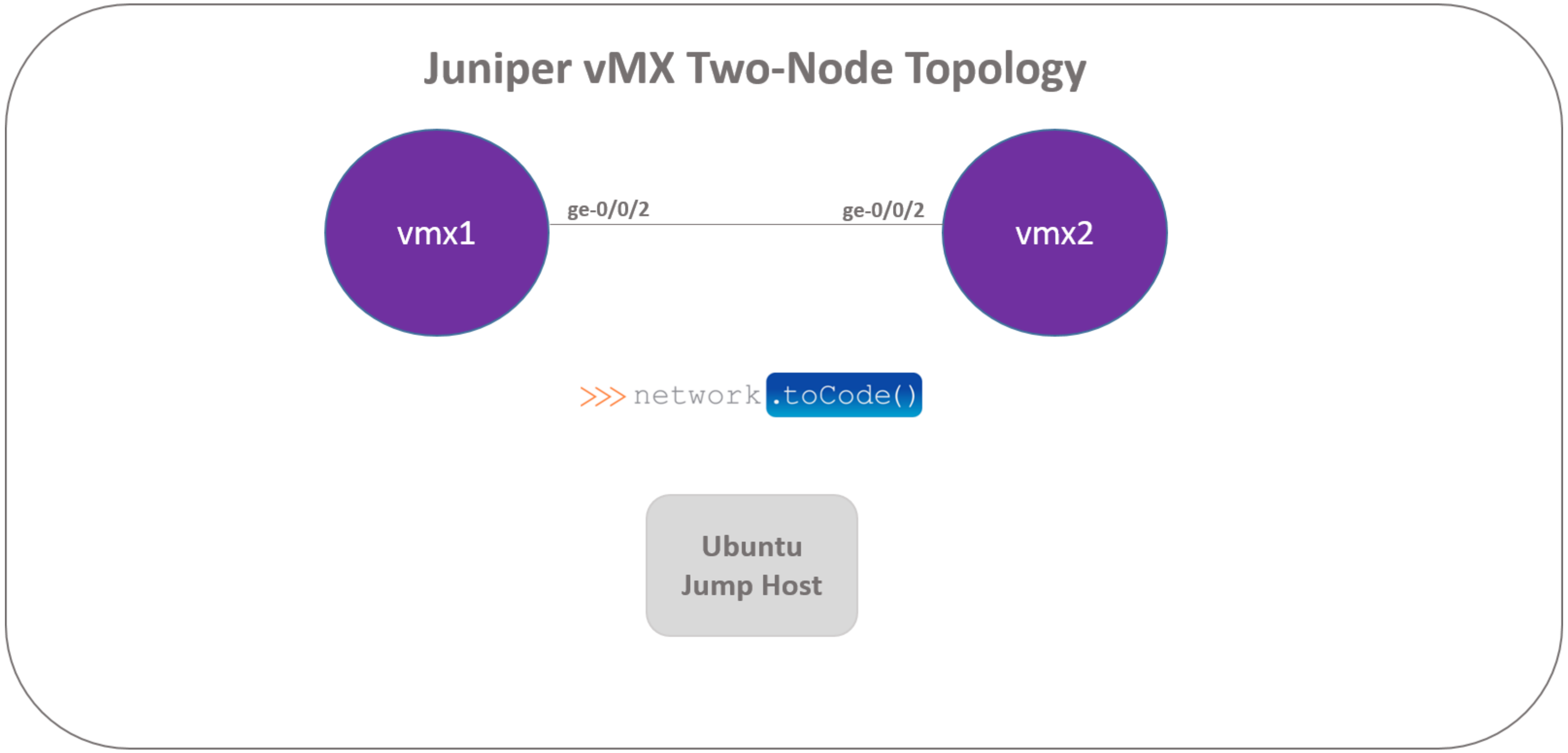
Diagram Refresher



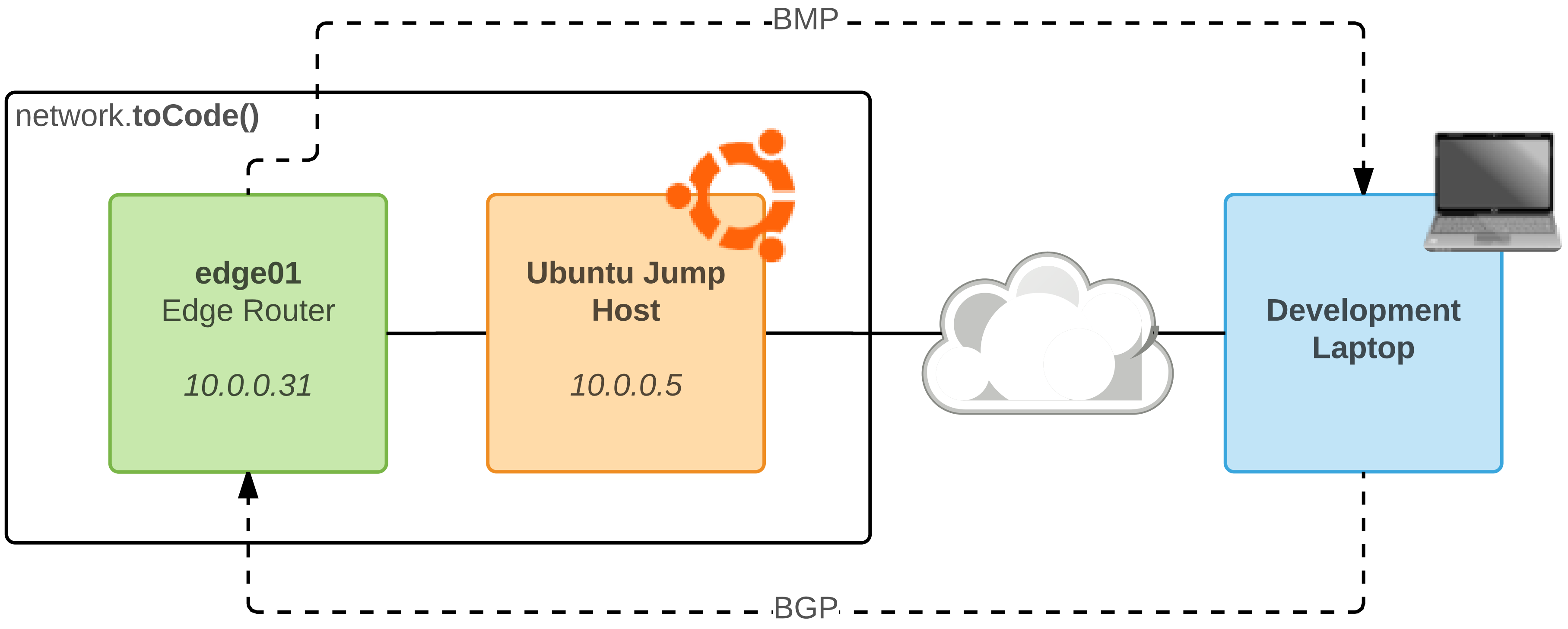
network.ToCode()

Two vMX lab

- vmx1 = our edge router (edge01)
- vmx2 = Both transit routers (transit01, transit02)



Developing on our laptop



Tunneling our development traffic

We need a way to connect from our laptop to edge01 for BGP as well as have edge01 connect to a BMP server running on our laptop.

SSH to the rescue

- Local forwarding `-L <local port>:<remote ip>:<remote port>`
- Remote forwarding `-R <port on jump host>:<local ip>:<local port>`

```
ssh -L 8179:10.0.0.31:179 -R :11019:localhost:11019 ntc@<jump host>
```

SSHD config

```
$ sudo vim /etc/ssh/sshd_config
Add GatewayPorts yes
```

TOP
SECRET

GoBGP

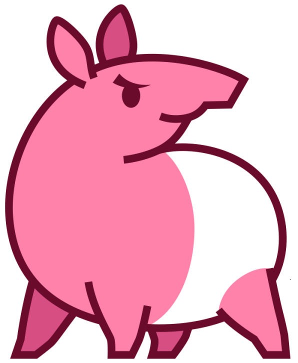
GoBGP

Go BGP is both a stand alone server (with RPC API!) written in Go. You can interact with it via command line or via GRPC.

- Webpage: osrg.github.io/gobgp/ (https://osrg.github.io/gobgp/)
- Github: github.com/osrg/gobgp (https://github.com/osrg/gobgp)

Installing

```
$ go get github.com/osrg/gobgp/gobgpd
$ go get github.com/osrg/gobgp/gobgp
```



Starting gobgpd

gobgpd is a standalone BGP daemon that needs no additional coding to run.

- Config files can be in TOML, YAML, or JSON

Starting the daemon :

```
$ gobgpd -f gobgpd.conf -p
```

Check the status with the gobgp command command

```
$ gobgp neighbors
$ gobgp global rib
```

Traffic engineer from from the command line

The gobgp can also allow use to originate routes from the BGP server to send out to the peer routers (edge routers)

Lets program in a host route to go bypass the shortest ASPATH

```
$ gobgp global rib add -a ipv4 198.51.100.50.0/24 nexthop 192.0.2.2
```

Modifying BGP from code

We have three options to interact with GoBGP from Go.

1. Run gobgpd standalone and interact with it via GRPC

2. Gun the gobgpd server in our own code base

3. Shell out to gobgp `-j` and parse JSON

BGP Monitoring Protocol

Relay BGP information back to a BMP server (monitoring station)

- RFC7854 (https://tools.ietf.org/html/rfc7854)
- Neighbor changes (peer up, peer down)
- Route updates
- Route mirroring
- Statistics

Let's write a BMP server to collect stats

Hack ideas

Peer monitor

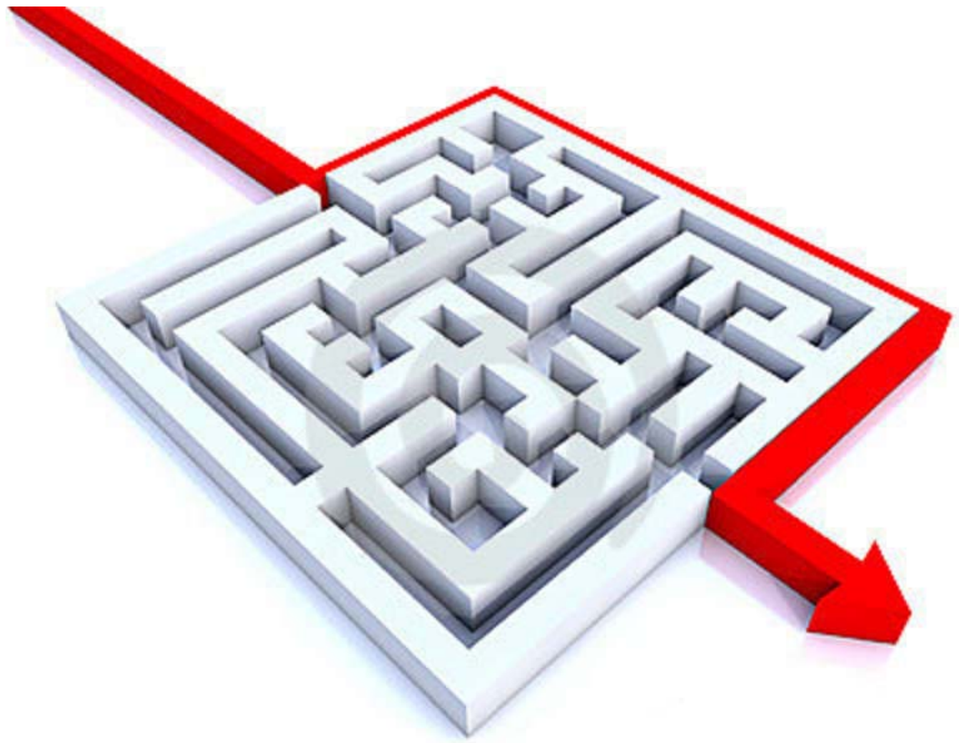
Use BMP to gather stats on BGP peers and routes they advertise overtime

- Setup BMP server and listen for all pre-policy events
- Setup alarms when peers go down
- Record stats on reachability, peer availability, withdrawals, into InfluxDB or other backend system.

BGP Optimal Path

Use BGP to inject routes to engineer around traffic issues, latency or even constants

- Find IP used by your application (netflow or application)
- Issue pings from router for each IP
- Program a route via BGP to the next-hop with the lowest latency and/or packet loss



Peer Egress Utilization Optimization

Run peers FLAMMING HOT!

- Monitor interface utilization (SNMP, netconf, stream telemetry)
- Gather netflow data
- When a certain threshold is reached program routes for new IPs onto transit

Conditional Server VIPs from BGP

- Run gobgp on a server
- Peer with upstream switches
- If a service is up advertise a bgp route
- If a service goes down withdrawal the bgp route
- bgp multipath == FREE LOAD BALANCING *

???

What ideas do you have?

Thank you

Brandon Bennett
Network Robot Mechanic, Facebook
bbennett@fb.com (mailto:bbennett@fb.com)
@brandonrbennett (http://twitter.com/brandonrbennett)