Global Foundation Services

DATA CENTERS NETWORKS SERVERS ENERGY SOFTWARE SECURITY

Microsoft

Brain-Slug: a BGP-Only SDN for Large-Scale Data-Centers

Adel Abouchaev, Tim LaBerge,
Petr Lapukhov, Edet Nkposong

Presentation Outline

- Problem Overview
- SDN Controller Design
- Handling Failure Scenarios
- Feature Roadmap
- Conclusions

Problem Overview

BGP-Only Data Centers

Presented at NANOG55

BGP is the better IGP!

Clos topology for network fabric

100K bare metal servers and more!

IETF draft available

Equal-Cost Multipath

Layer 3 Only (no VLANs spanning)

Simple oblivious routing

Large fan-outs for big data-centers (32+)

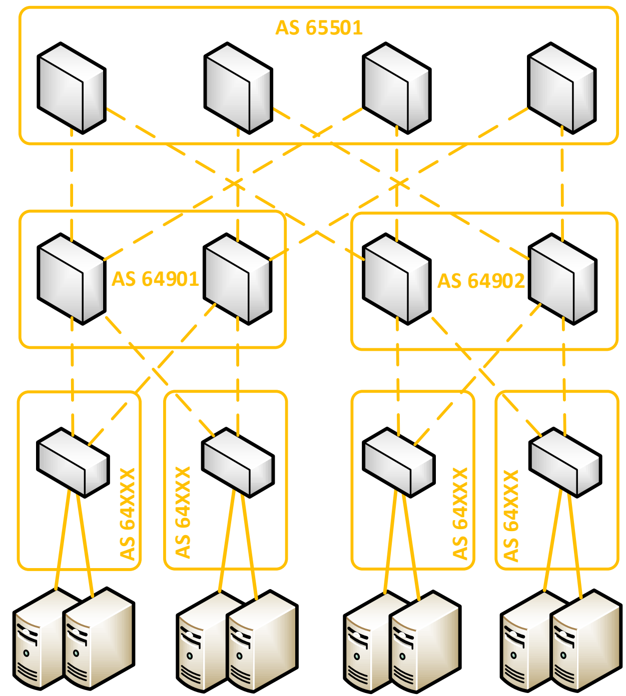
100% BGP Network

Simpler than IGP, less issues

Converges fast enough if tuned

Single protocol drives simplicity

Operated for more than 2 years now



SDN Use Cases for Data-Center

Not as many as one may think

First of all, SDN is defined as '*centralized packet forwarding control*'

But web-Scale data-centers are in fact **simple**

A lot of advanced logic is pushed to the servers already

- E.g. virtual overlays built in software (if any)
- Traffic filtering, stats collection and analysis
- Load-balancing and mobility

Network is simplified as much as possible

What problems are left in the network?

Changing routing behavior – but why?

…Remember routing logic is plain ECMP…

Still forwarding behavior needs to change sometimes

SDN Use Cases for Data-Center

Injecting ECMP Anycast prefixes

Used for load-balancing in the network

Or to provide resiliency across the WAN

Changing ECMP traffic proportions

Unequal-cost load distribution in the network

E.g. to compensate for various link failures and re-balance traffic

More generic traffic engineering for arbitrary topologies

Moving Traffic On/Off of Links/Devices

Without using any sort of CLI intervention/expect script etc

Strive for zero packet loss, graceful 'shutdown'

Multiple uses for this simple operation

- Graceful reload and automated maintenance
- Automated Isolation of network issues in "black box" scenarios

Goals and Non-Goals of project

Goals

- Deploy on existing networks, without software upgrades
- Low risk deployment, should have easy rollback story
- Leverage existing protocols/functionality
- Override **some** routing behavior, but keep non-SDN paths where possible

Non-Goals

- Network virtualization or segmentation, etc
- Per-flow, highly granular traffic control
- Support for non-routed traffic (e.g. L2 VPN)
- Optimizing existing protocols or inventing new ones
- Full control over all aspects of network behavior (low-level)

SDN Controller Design

Network Setup

Device Configuration

New configuration added

- Template to peer with the central controller (passive listening)
- Policy to **prefer** routes injected from controller
- Policy to announce only **certain** routes to the controller

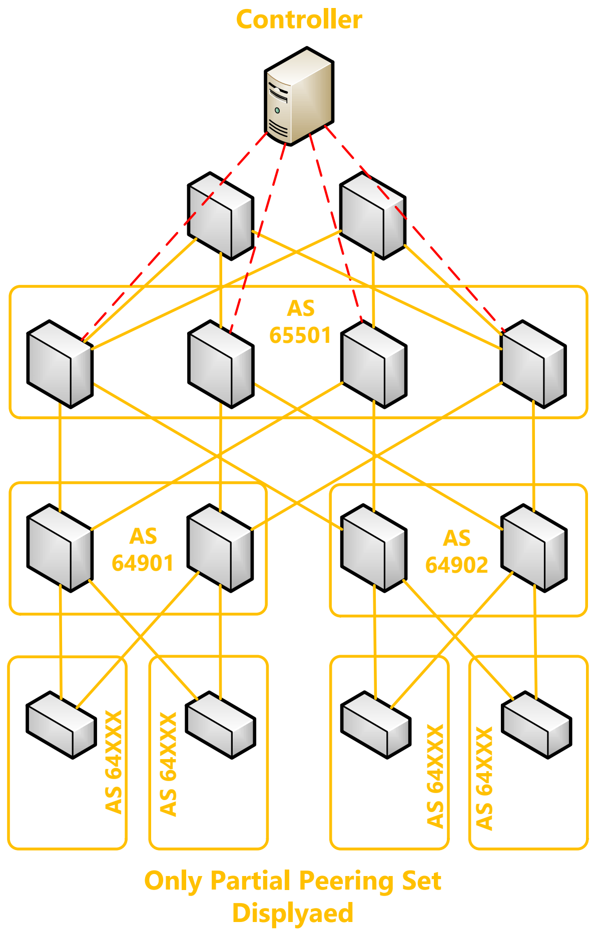
Peering to the Controller

Peering with all devices: multi-hop

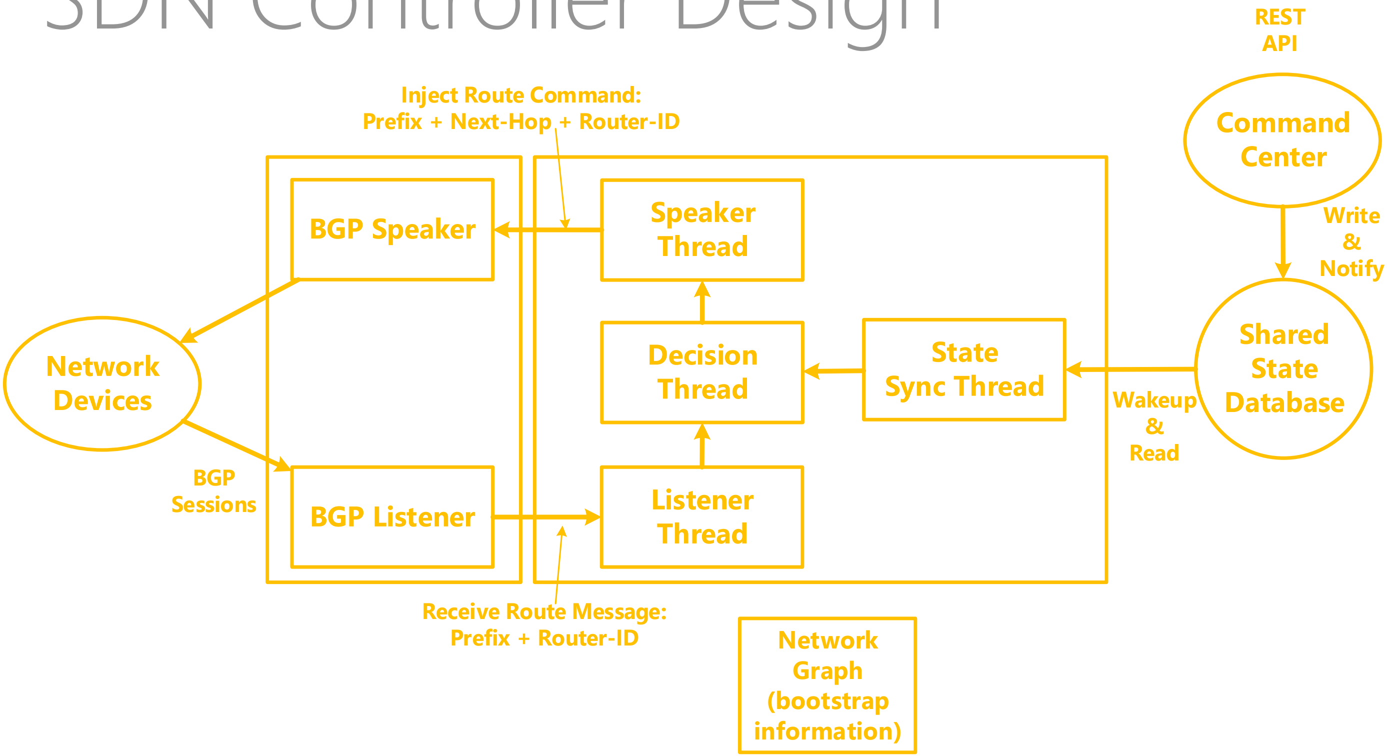
Key requirement: **path resiliency**

Clos has very rich path set

Network partition is highly unlikely



SDN Controller Design



- BGP Speaker/Listener(s) could scale horizontally (no shared state)
- Controller stores link-state of the network (next slides)
- Shared state could be anything e.g. devices to bypass/overload

BGP Speaker/Listener

Simplified functionality

Does not need to perform best-path selection

Does not need to relay BGP updates

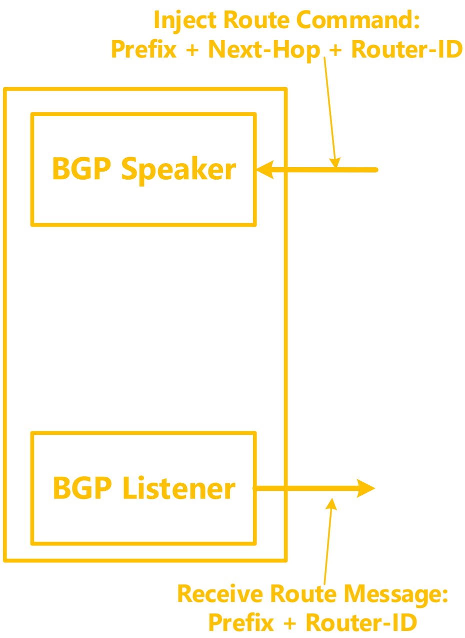
API

BGP Listener [stateless]

- Tell controller of prefixes received
- Tell controller of BGP sessions coming up/down
- Preferably using structured envelope (JSON/XML)

BGP Speaker [stateful]

- API to announce/withdraw a route **to a peer**
- Keep state of announced prefixes
- Note: State not shared among speakers



Implementation

Current P.O.C uses open-source **ExaBGP**

Implementing a simple C# version

Building Network Link State

Link-State discovery via BGP

Use a simple form of **control plane ping**

BGP session reflects physical link health

- Assumes *single* BGP session b/w two devices
- Could be always achieved using port-channels

How it works

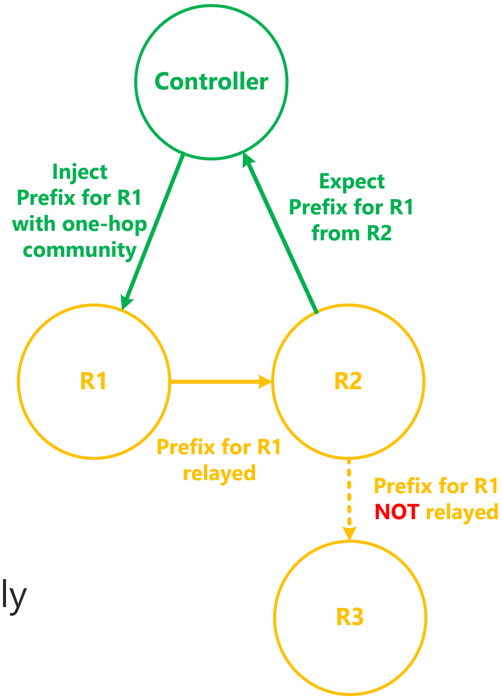
Create a **/32 host route** for every device

Inject prefix for device X into device X

Expect to hear this prefix via all devices $\mathbf{Y_1 \ldots Y_n}$, directly connected to X

If heard, declare link between X and Y as up

Community tagging + policy ensures prefix only leaks "one hop" from point of injection



Overriding Routing Decisions

Populating Routing Tables

The controller knows of all “edge” subnets and devices (e.g. ToRs)

Run SPF and compute next-hops (BFS works in most cases, O(m))

- For every “edge” prefix at every device
- Check if this is different from “baseline network graph” decisions
- Only push the “deltas”
- Prefixes are pushed with **third party next-hops** (next slide)

Key observations

Zero delta if no differentce from “baseline” routing behavior

Controller may declare a link/device down to re-route traffic

Implements the “overload” functionality

Overriding Routing Decisions

What about next-hops?

Injected routes have third-party next-hop

Those need to be resolved via BGP

Next-hops have to be injected as well

A next-hop /32 is created for every device

Same **one hop** BGP community used

Same “keep-alive” prefix could be used as NH

Injecting ECMP Routes

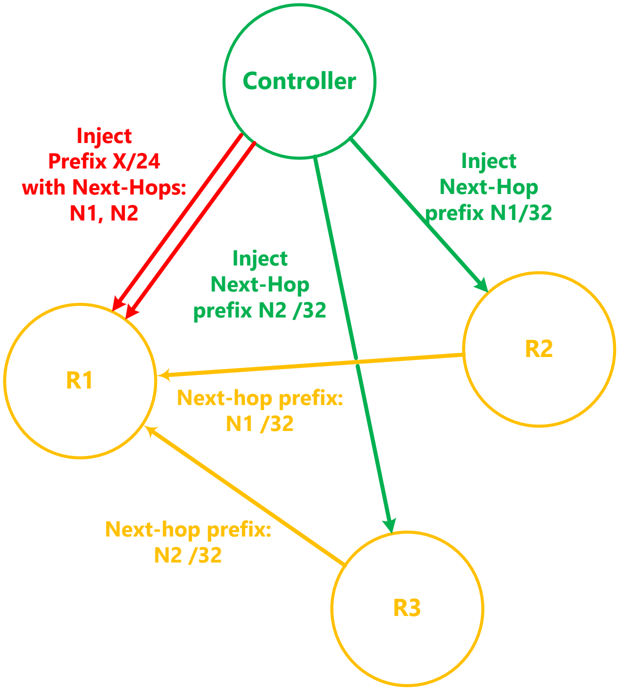
Only **one path** allowed path per BGP session

Need either Add-Path or multiple peering sessions

Worst case: # sessions = ECMP fan-out

Add-Path **Receive-Only** would help!

But model works in either case



Overriding Routing Decisions

How does it look on API side?

Simple REST to manipulate network state “overrides”

List of the supported calls:

- Logically shutdown/un-shutdown a link
- Logically shutdown/un-shutdown a device
- Announce a prefix with next-hop set via a device
- Read current state of the down links/devices etc

PUT http://…/overload/link/add=R1,R2&remove=R3,R4
PUT http://…/overload/router/add=R1&remove=R4

This requires a state database

State is **persistent** across controller reboots

State is **shared** across multiple controllers

State = overloaded links/devices, “static” prefixes

Ordered FIB Programming

Distributed programming poses issues

If updating BGP RIB's on devices in no particular order

RIB/FIB tables could go out of sync for some time

Well-known *micro-loops* problem!

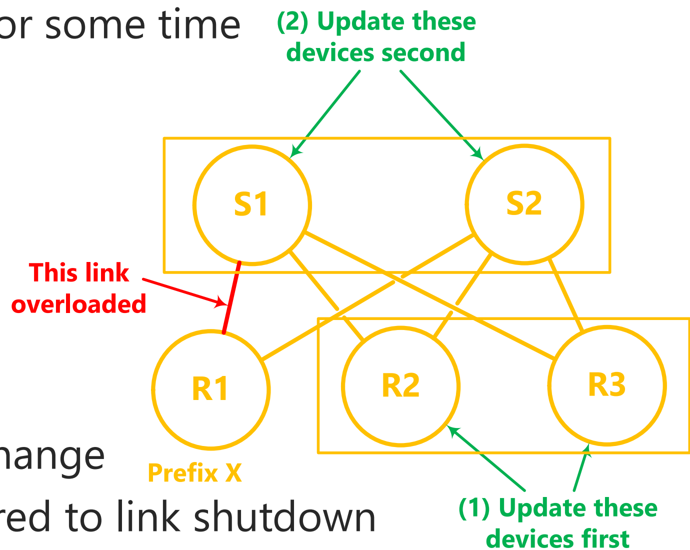
Central controller helps

For every link state change

- Build reverse-SPF for link event
- Update prefixes from leafs to root
- Controller sequences the updates

The updates "implode" toward the change

Packet loss 40-50x times less compared to link shutdown



Note: This logic assumes FIB programming is "reasonably" fast!

Handling Failure Scenarios

Handling Network Failures

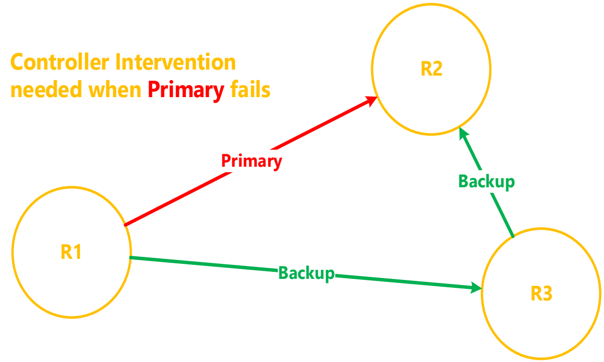
Physical network failures

Controller may add convergence overhead

- Only if prefix is *controlled* by BGP SDN
- …And if no backup paths available locally!

Convergence delay includes

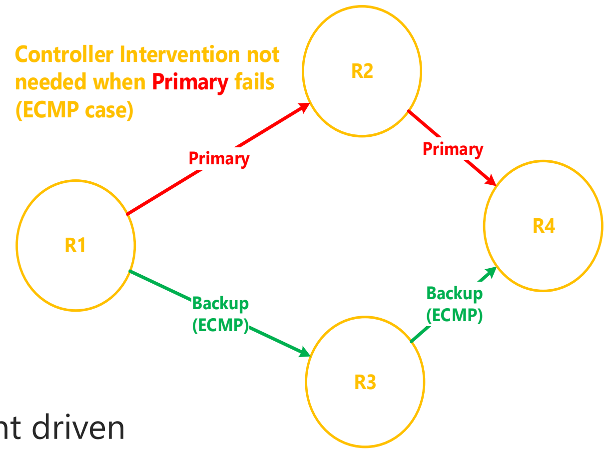
- Detecting link fault/BGP session shutdown
- Withdrawing the “keep-alive” prefix
- Controller finding new next-hop
- Controller pushing new path information
- Measured to be < 1 second



ECMP to the rescue!

In many cases, backup paths are available

- E.g. if an uplink fails, BGP re-converges locally
- Requires that BGP recursive resolution be event driven
- Possible to do local restoration in FIB agent



Multiple Controllers

Implement 1+N redundancy

Run N+1 controllers in “all active” fashion

Any single controller may command the network

Inject routes with different MED’s according to *priority*

- Higher MED paths used as backup

This way, backup routes are always in BGP table!

State sharing among controllers

Need to share the “overloaded” link/device information

In-memory database, replicated using Paxos algorithm

P.O.C. done via ConCoord objects (OpenReplica)

ConCoord also used for inter-process coordination

- Shared database locking
- Notifying controllers of state change

Multiple Controllers cont.

Controller bootstrap process

Discovers the following from configuration files:

- Static network topology
- Prefixes bound to “edge” devices
- IP addresses to peer with
- Controller’s **priority**

Obtains Concoord event object for synchronization

Reads the shared state from ConCoord object (R/W lock)

Starts peering sessions with all devices

- Expects X% of all known links to be up before making decisions

Injects “override” routes with MED=100+priority

Assumes the reconstructed network state is *eventually consistent* across the controllers

Feature Roadmap

Multiple Topologies

Logical Networks

Subset of “edge” prefixes mapped to a separate logical topology

API to create topologies/assign prefixes to topologies

API to overload links/devices per topology

Computing Paths

Separate “overloaded” links/devices per topology

Independent SPF runs per topology

Physical fault report raised to all topologies

Use cases

Re-routing subset of traffic, as opposed to all traffic

E.g. for automated fault isolation process

Traffic Engineering

ECMP Routing is oblivious

Failures may cause traffic imbalances
This includes:

- Physical failures
- Logical link/device overloading

Central controller helps

Compute new traffic distribution

Program weighted ECMP

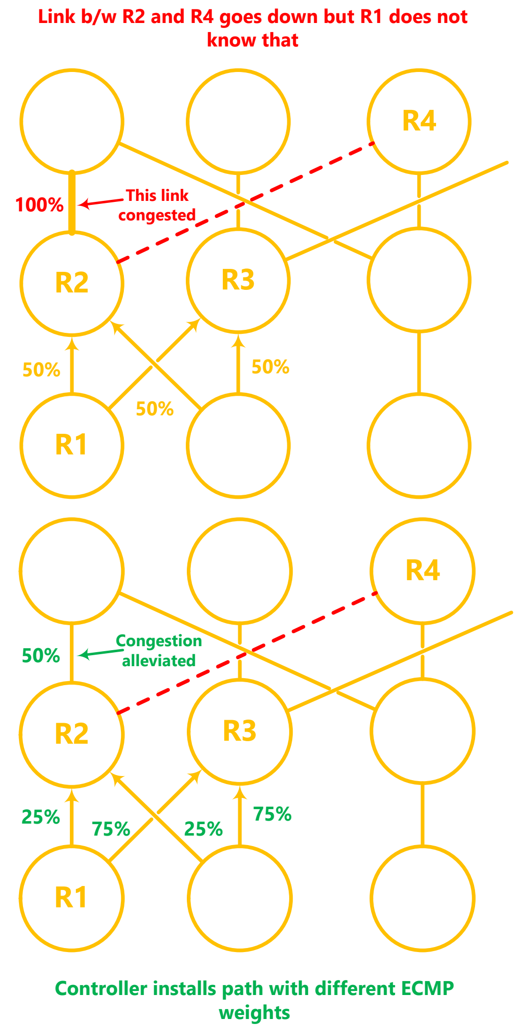
Signal using **BGP Link Bandwidth**

Not implemented by most vendors ☹

Generic Topologies

Not just Clos/Tree topologies

Think of BGP ASN as logical router



Traffic Engineering (cont.)

Implementation

Requires knowing the following

- Traffic matrix (TM)
- Network topology and link capacities

Solves Linear Programming problem

Compute & push **ECMP weights**

- For every prefix
- At every hop

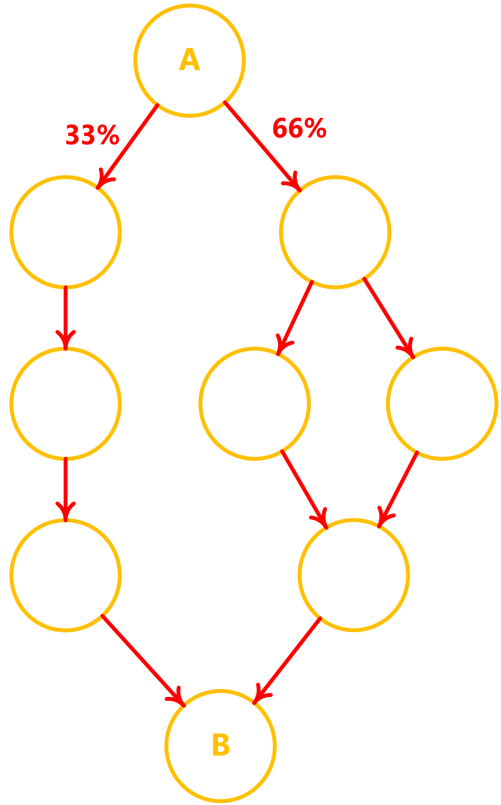
Optimal for a given TM

This has implications

Link state change causes reprogramming

More state pushed down to the network

More prefixes are now controlled



Ask to the vendors!

Weighted ECMP in DC switches

Most common HW platforms **can do it** (BRCM)

Signaling via BGP does not look complicated either

Note: Has implications on hardware resource usage

ECMP Consistent Hashing

Localized impact upon ECMP group change

Goes naturally with weighted ECMP

Well defined in RFC 2992 (ECMP case)

BGP Add-Path

Not a standard (sigh)

We'd really like to have receive-only functionality

Conclusions

Lessons learned

Be realistic in what you want

Clearly define use cases, don’t look for silver bullets

Operational implications in front of everything else

Ease of deployment is important

BGP does not require new firmware or API’s

Some BGP extensions are nice to have

Leveraging BGP makes life simple

BGP code is pretty mature (for most vendors)

Easy to **fail-back** to regular routing

Controller code is **very lightweight** (<1000 LoC)

BGP SDN is not universal but practical

Solves our current problems and in future allows for much more

References

BGP Routing for Data-Centers

http://datatracker.ietf.org/doc/draft-lapukhov-bgp-routing-large-dc/

Routing Control Platform

http://www.cs.princeton.edu/~jrex/papers/rcp-nsdi.pdf

ExaBGP

http://code.google.com/p/exabgp/

BGP Link-Bandwidth

http://datatracker.ietf.org/doc/draft-ietf-idr-link-bandwidth/

ConCoord/Openreplica

http://openreplica.org/

http://www.cs.cornell.edu/courses/cs7412/2011sp/paxos.pdf

Traffic Engineering with weighted ECMP

http://www.retitlc.polito.it/mellia/corsi/05-06/Ottimizzazione/744.pdf

http://www.seas.upenn.edu/~guerin/Publications/split_hops-Dec19.pdf