Experiences with BGP in Large Scale Data Centers: Teaching an old protocol new tricks

Presented by: Edet Nkposong, Tim LaBerge, Naoki Kitajima

Global Networking Services Team, Global Foundation Services, Microsoft Corporation

Agenda

- Network design requirements
- Protocol selection: BGP vs IGP
- Routing design detailed
- Motivation for BGP SDN
- Design of BGP SDN Controller
- Fault tolerance in BGP SDN
- The roadmap for BGP SDN

Design Requirements

Scale of the data-center network:

- 100K+ bare metal servers
- Over 3K network switches per DC

Applications:

- Map/Reduce: Social Media, Web Index and Targeted Advertising
- Public Cloud Computing: Elastic Compute and Storage
- Real-Time Analytics: Low latency computing leveraging distributed memory across discrete nodes.

Key outcome:

East ←→ West traffic profile drives need for large bisectional bandwidth.

Design Requirements

Cost Down

- Big Data: Storage of large data sets w/geographic replicas.
- Vendor Diversity: build ecosystem and drive competition
- Network CAPEX/OPEX is shifting:
  - From ASIC to Fiber and Optics.
  - From "big iron" to many dense commodity switches.
  - Power/cooling still important, especially closer to the core.

Site Up

- From "network availability" to "system availability"
- Design and Test for Failure (hardware does and will fail)
  - Systems have high component count (Processors, Disks, Optics, Cables)
  - Induce failure (up to failing an entire DC) to understand behavior of the system under varying conditions.

Translating Requirement to Design

Network Topology Criteria

Support East <-> West Traffic Profile

- Applications distributed over many servers
- Mice and elephant traffic flows

Provide Large Bisectional Bandwidth

- No over-subscription in Network Fabric
- Minimize Capex and Opex
  - Cheap commodity switches
  - Low power consumption

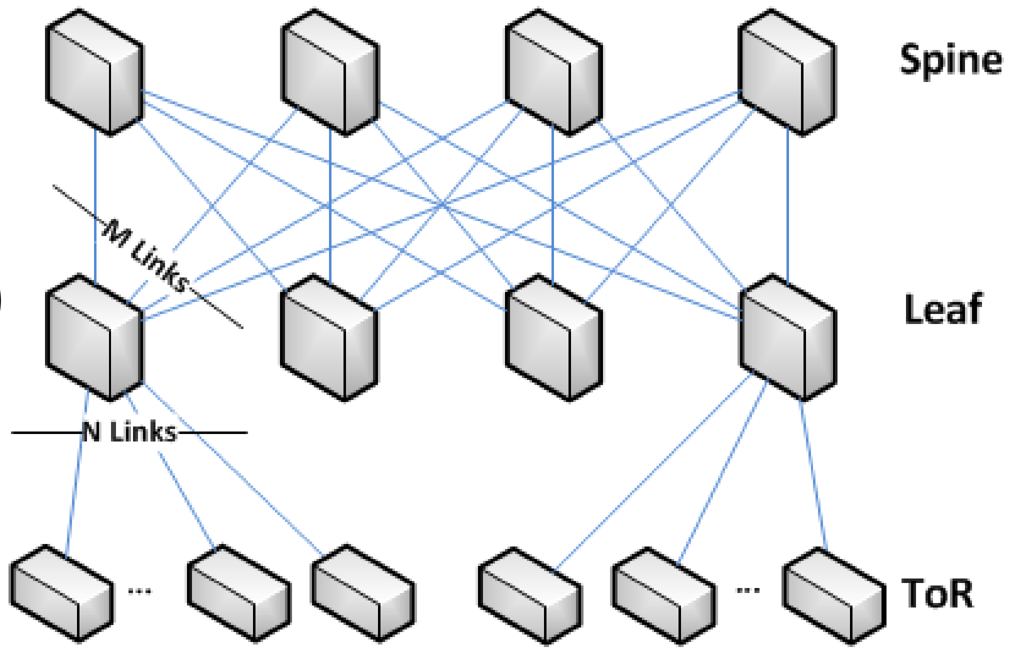
Use Homogenous Components

Network Protocol Criteria

- Standards Based
- Control Plane Scaling and convergence
- Minimize resource consumption e.g. CPU, TCAM usage - predictable and low
- Layer3 with equal-cost multipathing (ECMP)
- Programmable
  - Extensible and easy to automate

Network Design: Topology

- Example: 3-Stage Folded CLOS.
- Full bisection bandwidth (m ≥ n) .
- Horizontal Scaling (scale-out vs. scale-up)
- Natural ECMP link load-balancing.
- Viable with dense commodity hardware.
  - Build large “virtual” boxes out of small components



3-Stage Folded Clos Topology

Network Design: Protocol

Network Protocol Requirements

- Resilience and fault containment
  - CLOS has high link count, link failure is common, so limit fault propagation on link failure. [MSFT design does not hide link failures!]
  - Minimize effect of packet loss on application performance
- Control Plane Stability
  - Consider number of network devices, total number of links etc.
  - Minimize amount of control plane state.
  - Minimize churn at startup and upon link failure.
- Traffic Engineering
  - Heavy use of ECMP makes TE in DC not as important as in the WAN.
  - However we still want to “drain” devices and respond to imbalances

Network Design: Protocol

Operational Requirements

- Avoid Complexity: Configuration/troubleshooting.
- Predictability: Link failure should gracefully degrade capacity.
- Scalability: Can’t afford non-linearity in CPU/memory.
- Interoperability: It’s difficult to use vendor specific functionality.

Why BGP and not IGP?

- Simpler protocol design compared to IGPs
  - Mostly in terms of state replication process
  - Better vendor interoperability
  - Fewer state-machines, data-structures, etc
- Troubleshooting BGP is simpler
  - Paths propagated over link
  - ASPATH is easy to understand.
  - Easy to correlate sent & received state
- BGP allows for per-hop traffic engineering
  - Unique as compared to link-state protocols
  - Very helpful to implement granular policies
  - Use for unequal-cost Anycast load-balancing solution

Why BGP and not IGP? (cont.)

- Event propagation is more constrained in BGP
  - More stability due to reduced event “flooding” domains
  - E.g. can control BGP UPDATE using BGP ASNs to stop info from looping back
  - Generally is a result of distance-vector protocol nature
- Configuration complexity for BGP?
  - Not a problem with automated configuration generation
  - Especially in static environments such as data-center
- What about convergence properties?
  - Simple BGP policy helps.
  - Practical convergence in less than a second

Lessons from Route Surge PoC Tests

We simulated PoC tests using OSPF and BGP, details at end of Deck.

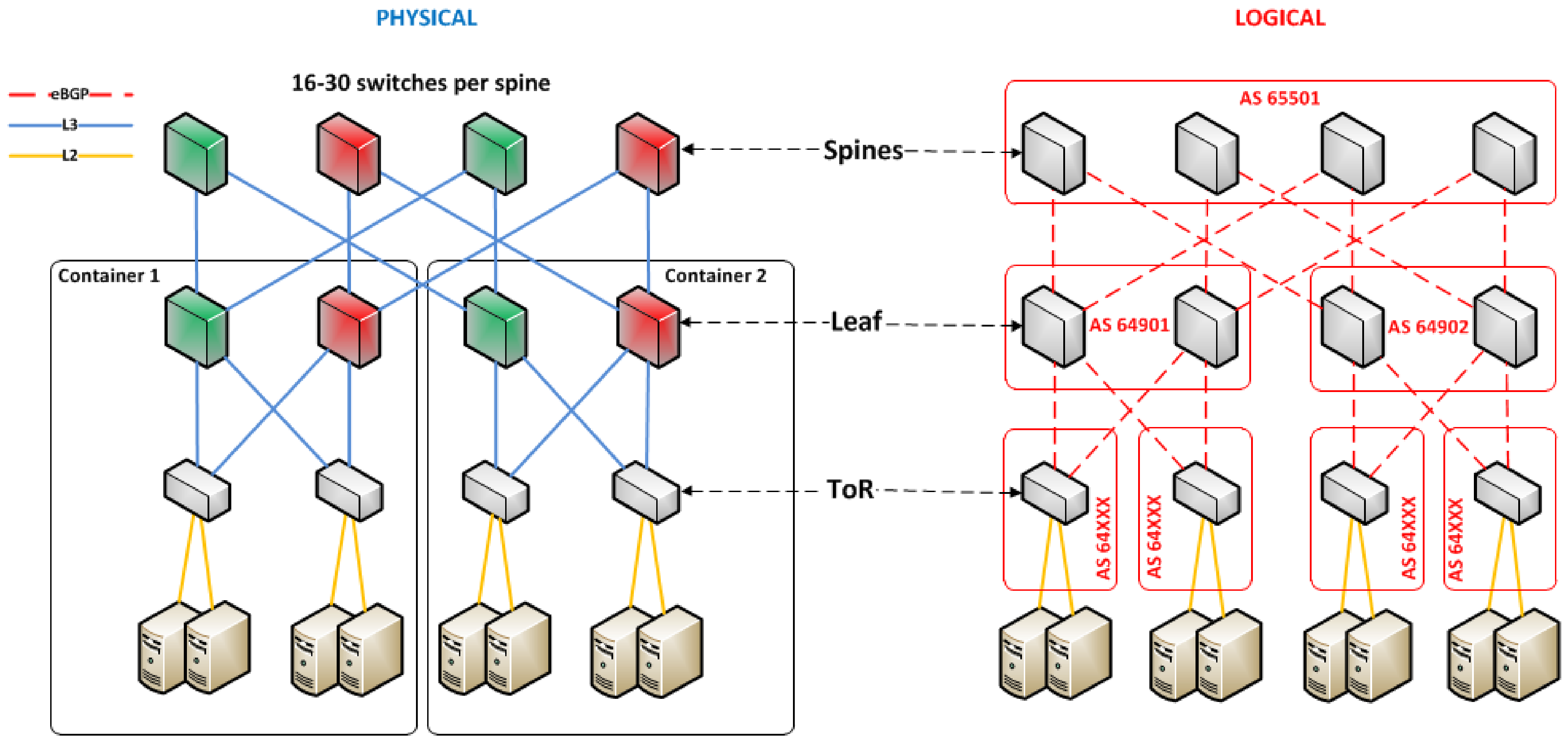
- Note: some issues were vendor specific ☺ Link-state protocols could be implemented **properly**!, but requires tuning.
- Idea is that LSDB has many "useless" non-best paths.
- On link failure, these "useless" paths are installed in fib.
- This results in a surge in FIB utilization---Game Over.
- With BGP, ASPATH keeps only "useful" paths---no surge.

Routing Design

- 3 Stage Folded CLOS Topology with parallel Spine Blocks
- PoC Tests prove BGP meets our design requirements
- Border Switch
  - Connects to the WAN
  - 4 or more switches in a unique ASN (64xxx)
- Spine
  - 4 Parallel Spine Blocks (multiple switches per block)
  - All Spine Blocks use a unique ASN (64xxx)
  - All Spines connect to all Border Switches
- Container
  - Comprised of multiple ToRs attached to 4 Leaf Switches
  - Each ToR (64xxx) has a unique ASN
  - ToR ASNs re-used across containers
  - All Leaf Switch use a unique ASN (64xxx)
  - Each Leaf Switch is attached to a unique spine block

Routing Design (contd)

- Single logical link between devices (portchannel).
- eBGP all the way down to the ToR.
- Separate BGP ASN per ToR.

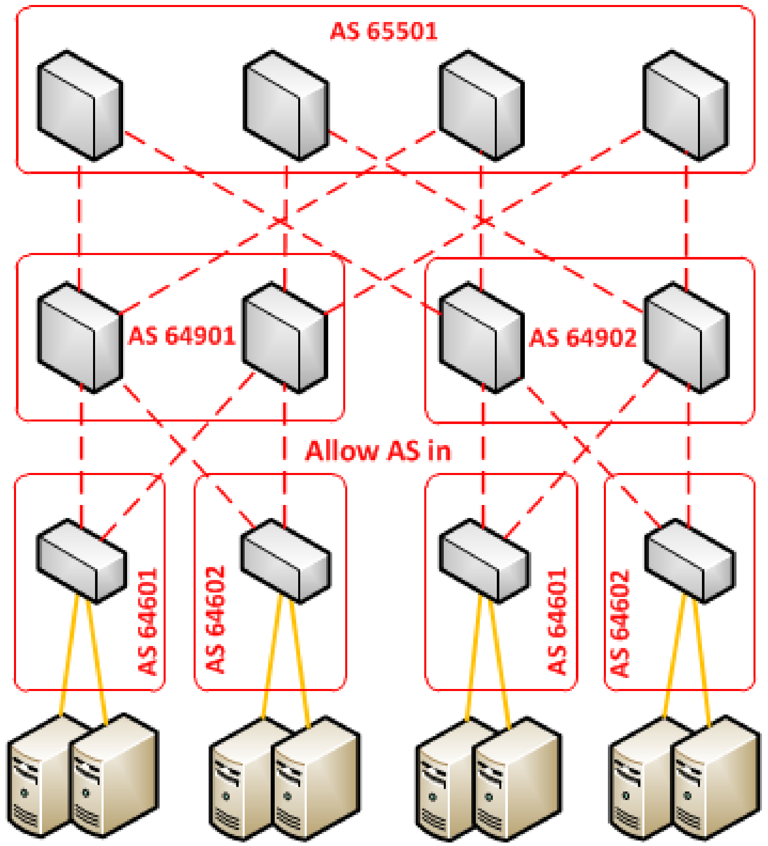


BGP Routing Design Specifics

- BGP AS_PATH Multipath Relax
  - For ECMP even if AS_PATH doesn’t match.
  - Sufficient to have the same AS_PATH length
- We use 2-octet private BGP ASN’s
  - Simplifies path hiding at WAN edge (**remove private AS**)
  - Simplifies route-filtering at WAN edge (single regex).
  - But we only have 1022 Private ASN’s…
- 4-octet ASNs would work, but not widely supported

BGP Specifics: Allow AS In

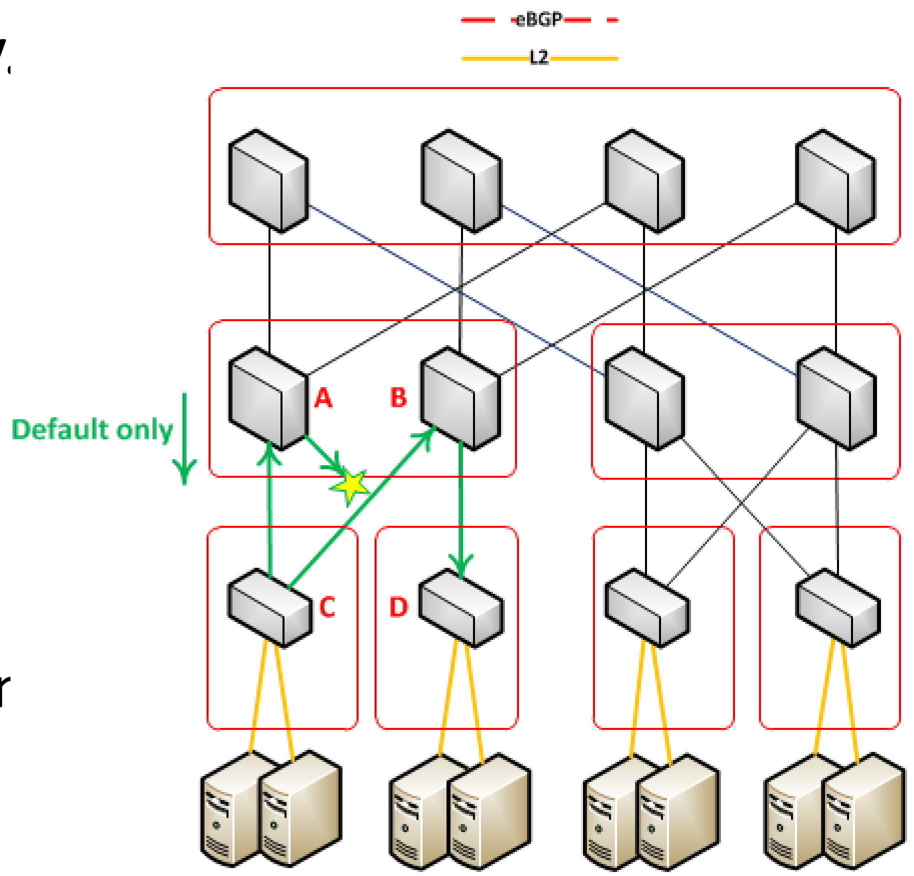
- This is a numbering problem: the amount of BGP 16-bit private ASN's is limited
- Solution: reuse Private ASNs on the ToRs.
- *"Allow AS in"* on ToR eBGP sessions.
- ToR numbering is local per container/cluster.
- *Requires vendor support, but feature is easy to implement*



AllowAS In configured on ToR eBGP sessions to the Leafs

BGP Specifics: Default Routing

- Default route for external destinations only.
- Don’t hide more specific prefixes.
- O.W. Route Black-Holing on link failure!
- If D advertises a prefix P, then some of the traffic from C to P will follow default to A. If the link AD fails, this traffic is black-holed.
- If A and B send P to C, then A withdraws P when lir AD fails, so C receives P only from B, so all traffic will take the link CB.
- **NOTE: could be solved with BGP SDN**



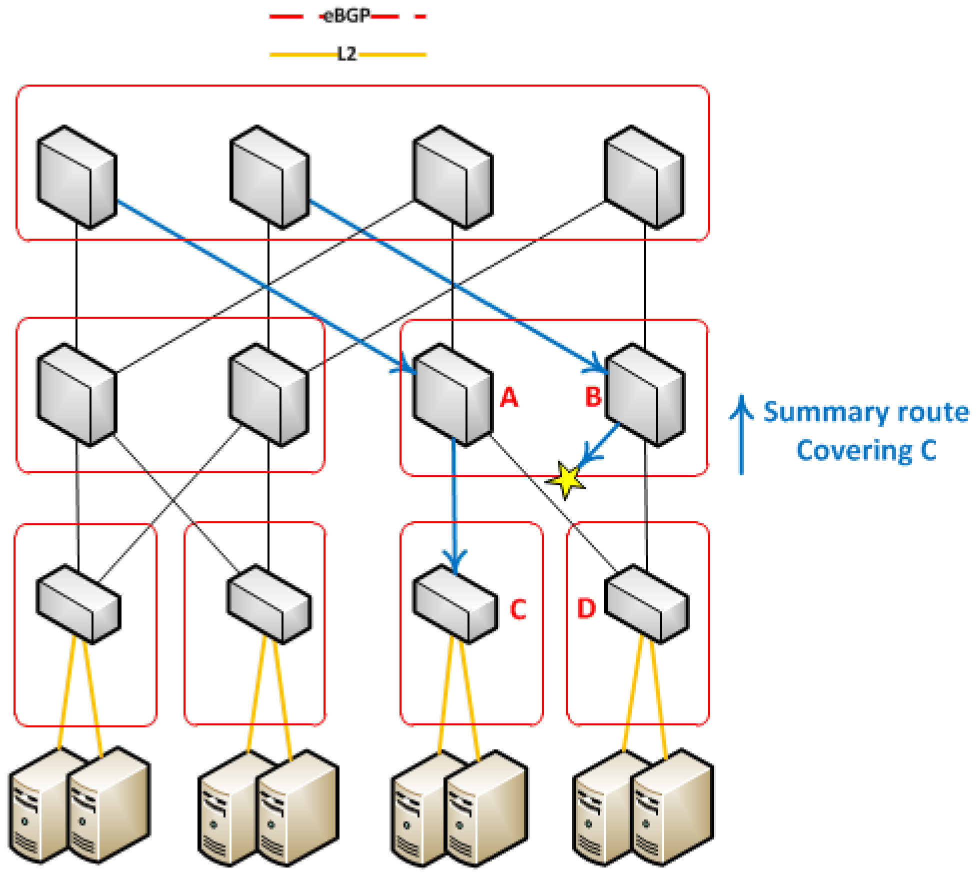
Using default route leads to blackholes

BGP Specifics: Route Summarization

- Don’t summarize server subnets!
- O.W., Route Black-Holing on link failure!
- Suppose C and D advertise prefixes that A and B summarize via a less specific prefix P.
- Some traffic sent to C will transit B. If link BC fails, this traffic is black-holed.
- Without summarization, B withdraws C’s prefix on failure of link BC, so all traffic for C transits A.

(Summarizing P2P links is OK.)

NOTE: black-holing problem be solved with BGP SDN!



Summarizing leads to route blackholing

Operational Issues with BGP

- Consistent feature support
  - Not all vendors support everything you need, e.g.:
  - BGP Add-Path
  - 32-bit ASNs
  - AS_PATH multipath relax
  - BGP update groups
- Interoperability issues:
  - Especially when coupled with CoPP and CPU queueing
  - Small mismatches may result in large outages!

Operational Issues with BGP

- Unexpected ‘default behavior’
  - E.g. selecting best-path using ‘oldest path’
  - Combined with lack of as-path multipath relax on neighbors…
- Traffic polarization due to hash function reuse
  - This is not a BGP problem but you see it all the time
- Overly aggressive timers – session flaps on heavy CPU load
- RIB/FIB inconsistencies
  - This is not a BGP problem but it is consistently seen in all implementations

The Next Step: BGP SDN for Data-Centers

Tim LaBerge, Edet Nkposong

Microsoft

Software Driven Networks - Data Center

- Programmability
  - Automation - deployment/configuration/monitoring concerned with scale, quality, consistency
    - Configuring large quantities of commodity hardware
  - Programming - extract value, new creative uses of the network, experimentation
- NFV promoting a shift away from dedicated appliances. e.g. virtual overlays/load balancing built in software
- Control routing behavior via Cloud Orchestration platform - tighter coupling with application components.
  - Path Isolation (route control)
  - Load Balancing
  - Data Center and WAN Network Integration

SDN Use Cases for Data-Center

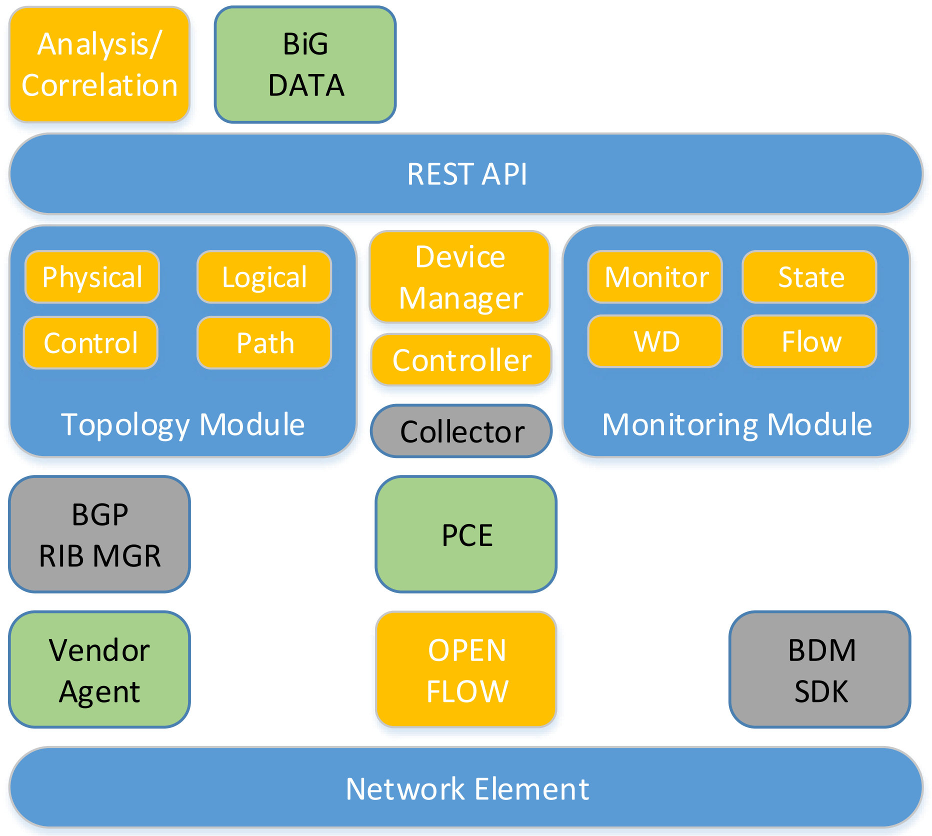
- Injecting ECMP Anycast prefixes
  - Used for load-balancing in the network
  - …Or to provide resiliency across the WAN
- Moving Traffic On/Off of Links/Devices
  - Strive for zero packet loss
  - Multiple uses for this simple operation
    - Graceful reload and automated maintenance
    - Isolating network issues in “black box” scenarios
- Changing ECMP traffic proportions
  - Unequal-cost load distribution in the network
  - E.g. to compensate for various link failures and re-balance traffic

BGP SDN Controller

- Focus is the DC – controllers scale within DC, partition by cluster, region and then global sync

- Controller Design Considerations
  - Logical vs Literal
  - Scale - Clustering
  - High Availability
  - Latency between controller and network element

- Components of a Controller
  - Topology discovery
  - Path Computation
  - Monitoring and Network State Discovery
  - REST API



Controller is a component of a Typical Software Orchestration Stack

BGP SDN Controller (contd.)

- **We do literal SDN – towards dynamically managing network state in software.**
  - Need a starting point ---> Graphs ground state
  - Need current running state ---> Link up/down? Hosts present?
  - Need to compute new desired state. <---- Only thing that's new.
  - Need to inject desired forwarding state.

- **Programming the Network via Indirect influence of the RIB**
  - Topology discovery via BGP for Links State Computation
  - Indirect influence via established BGP sessions to inject alternate policies, network paths etc.
  - Alternative methodology proposed in I2RS which will allows direct programming of RIB via RIB Manager. Still in draft.

- **How does this compare to OF?**
  - OpenFlow would allow programming of the FIB but requires adopting new Protocol.
  - Silicon that implements OF 1.1? 1.2? 1.3?
  - OS code/SDK to program silicon.
  - API's to expose the SDK to the control plane.
  - Controller.

SDN Controller Design

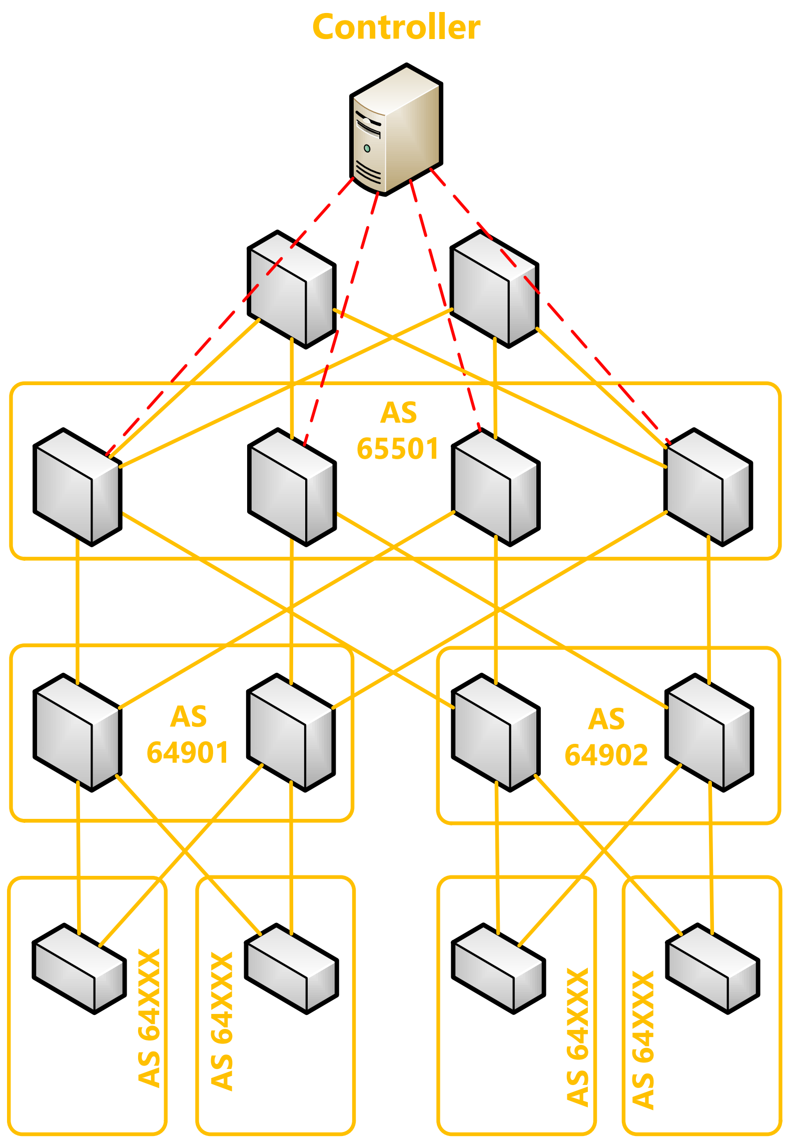
Network Setup

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

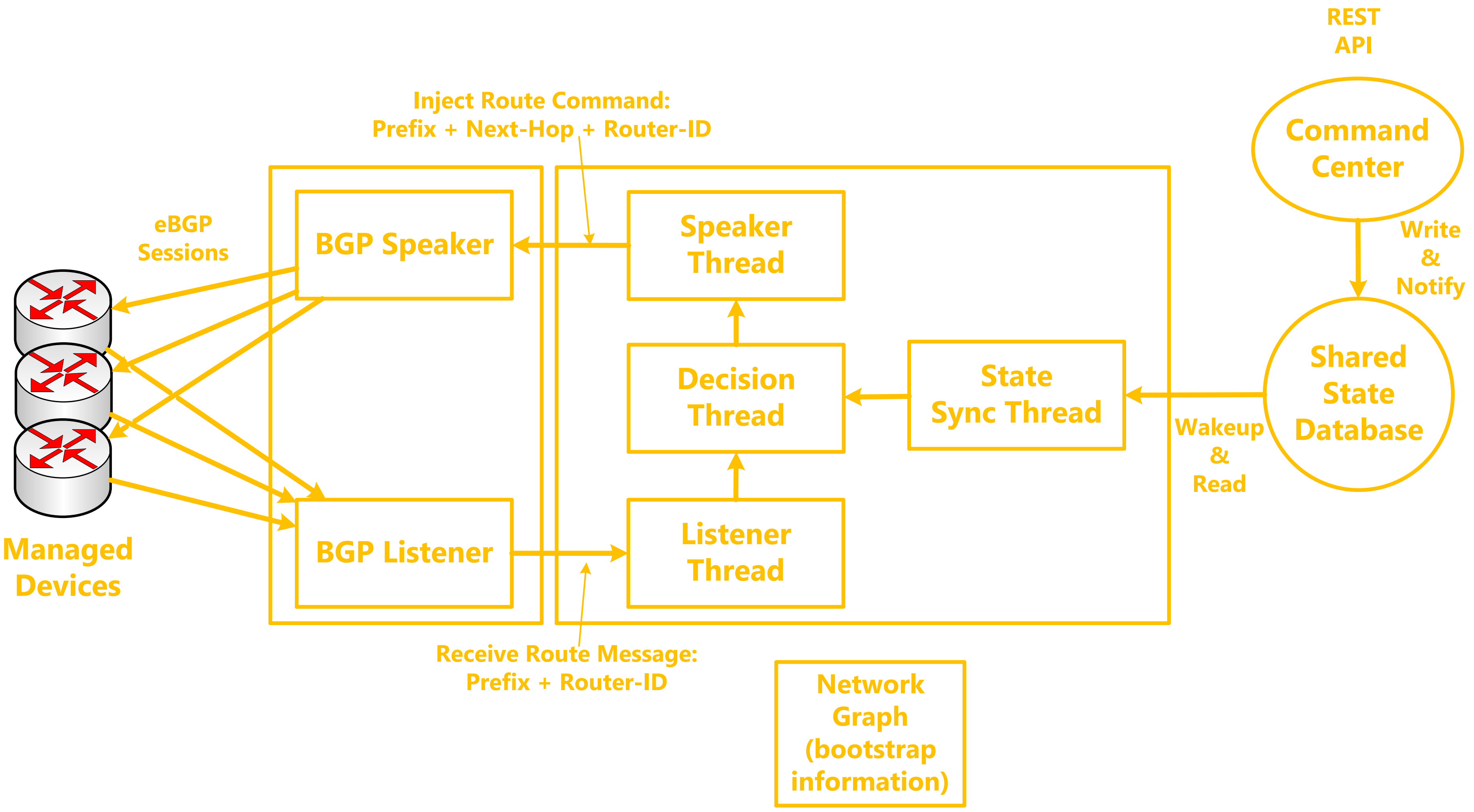
Peering to the Controller

- Peering with all devices: multi-hop
- Key requirement: path resiliency
- Clos has very rich path set
- Network partition is very unlikely



Only Partial Peering Set Displyaed

SDN Controller Design



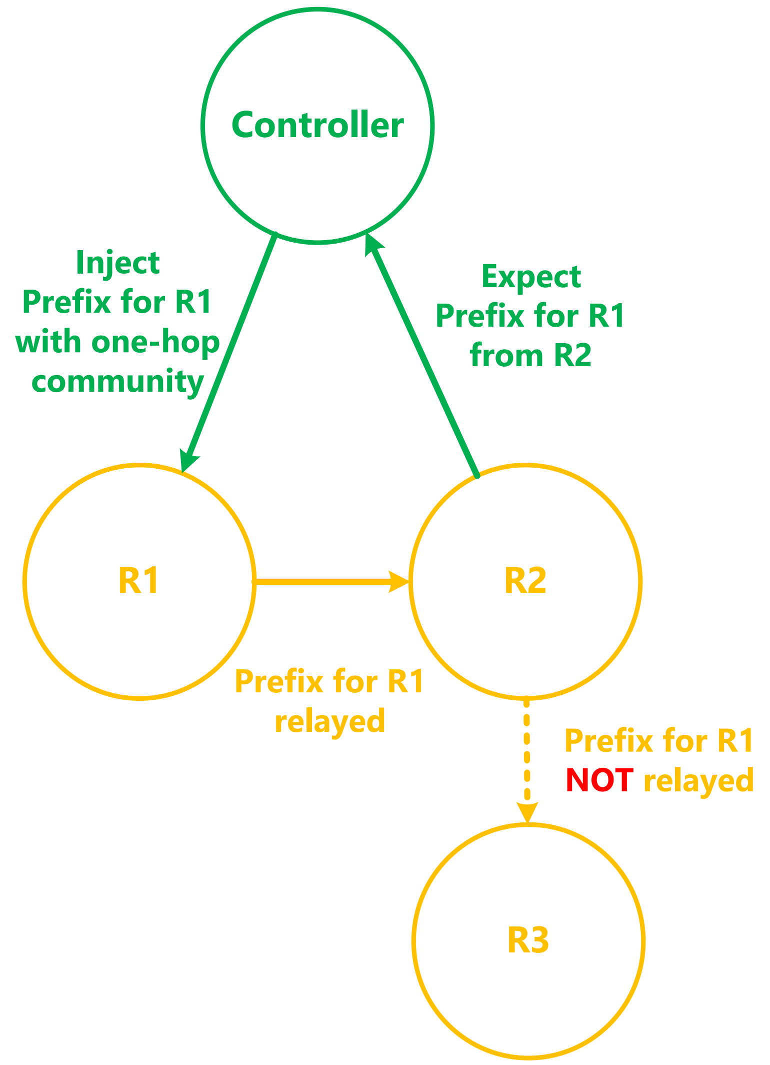
BGP Speaker/Listener

- 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
    - Keep state of announced prefixes
- Implementation
    - Implemented a C# version
    - P.O.C used ExaBGP

Building Network Link State

- Goal: Build Link-State of Live Network
  - Use a special form of "**control plane ping**"
  - Rely on the fact that BGP session reflects "link health"
  - Assumes single BGP session b/w two devices
- How it works
  - Create a /32 prefix for every device
  - Inject prefix for device X into device X
  - Expect to hear this prefix via all devices Y1…Yn, 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


Controller
Inject Prefix for R1 with one-hop community
Expect Prefix for R1 from R2
R1
R2
Prefix for R1 relayed
Prefix for R1 NOT relayed
R3

Overriding Routing Decisions

- Populating Forwarding Databases

The controller knows of all “edge” subnets and devices
Runs SPF and computes next-hops…

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

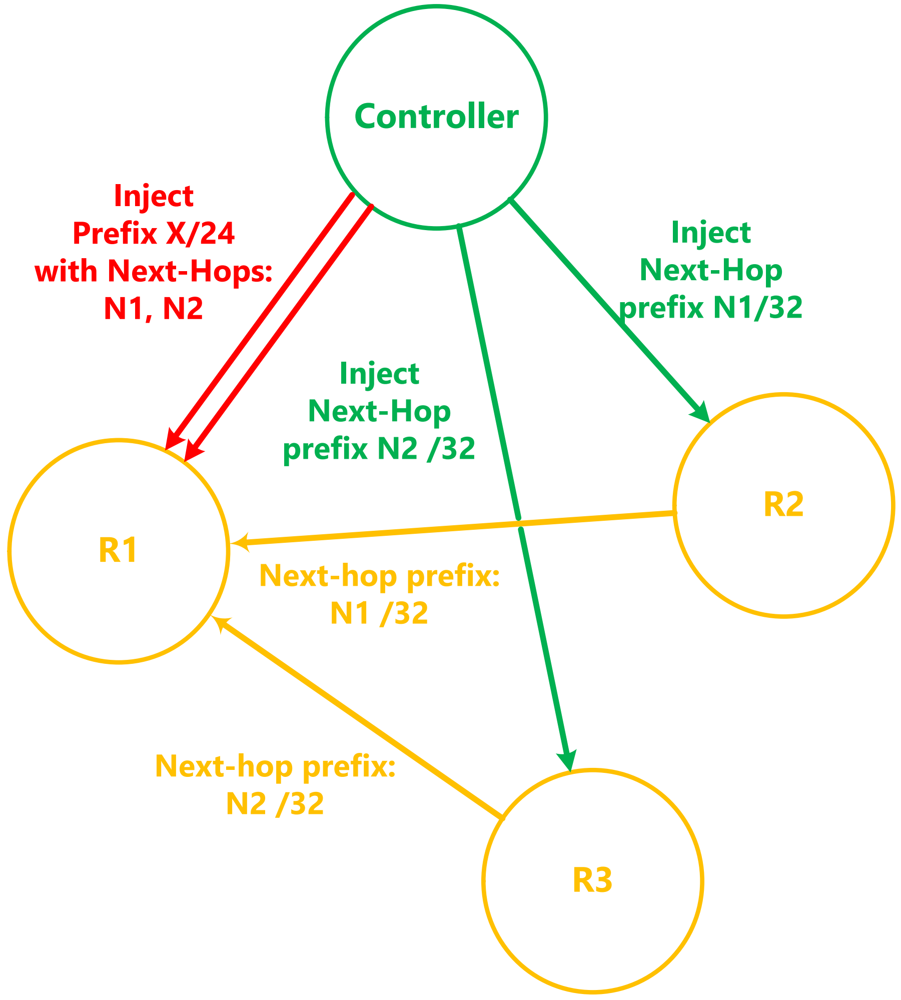
- Key observations
  - Controller has full view of the topology
  - Zero delta if no differentce from “default” routing behavior
  - Controller may declare a link down to re-route traffic…

  … Even if the link is physically up

This allows for seamlessly “overloading” links and devices

Overriding Routing Decisions cont.

- 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
- 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!


Controller
Inject Prefix X/24 with Next-Hops: N1, N2
Inject Next-Hop prefix N1/32
Inject Next-Hop prefix N2 /32
R1
R2
R3
Next-hop prefix: N1 /32
Next-hop prefix: N2 /32

Overriding Routing Decisions cont.

- 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

PUT http://<controller>/state/link/up=R1,R2&down=R3,R4

- This requires a state database
  - State is **persistent** across controller reboots
  - State is **shared** across multiple controllers

Ordered FIB Programming

- Distributed programming poses issues

  If updating BGP RIB's on devices in random order…

  …RIB/FIB tables could go out of sync

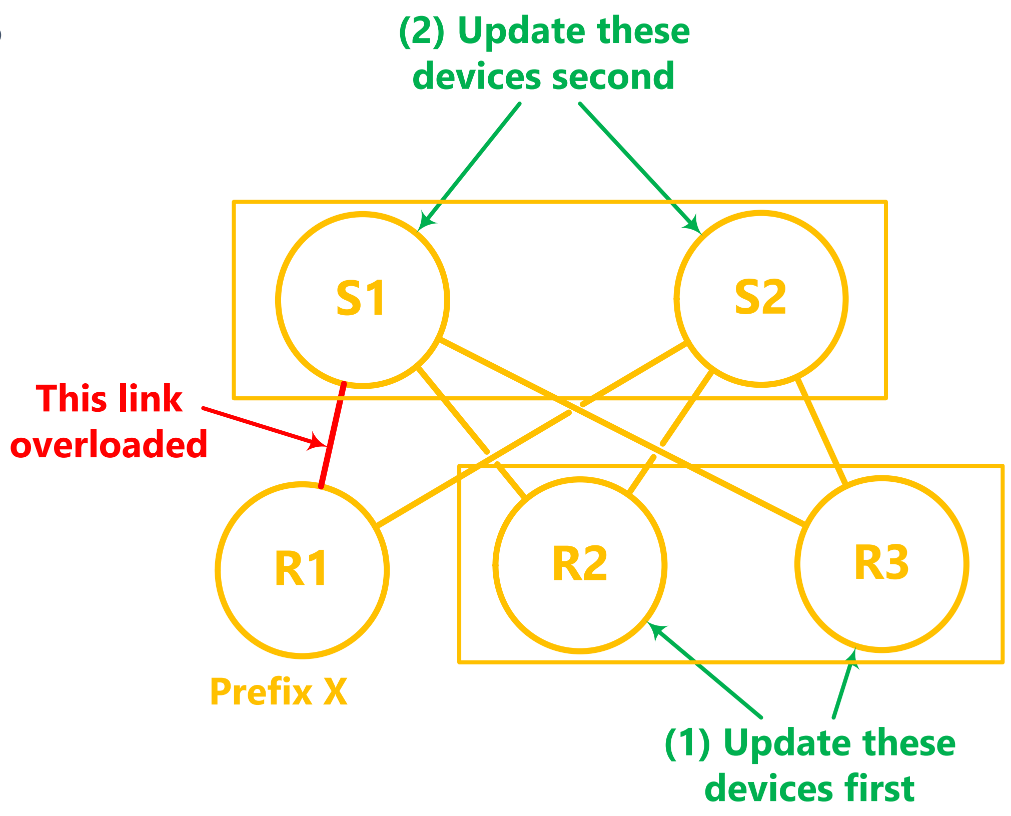
  Micro-loops problem!

- Central controller helps

  For every link state change

  - Find prefixes affected
  - Build reverse-SPF for each prefix
  - Update from the leafs to the root
  - Controller sequences the updates

Note: assumes FIB programming is fast!


(2) Update these devices second
S1
S2
This link overloaded
R1
R2
R3
Prefix X
(1) Update these devices first

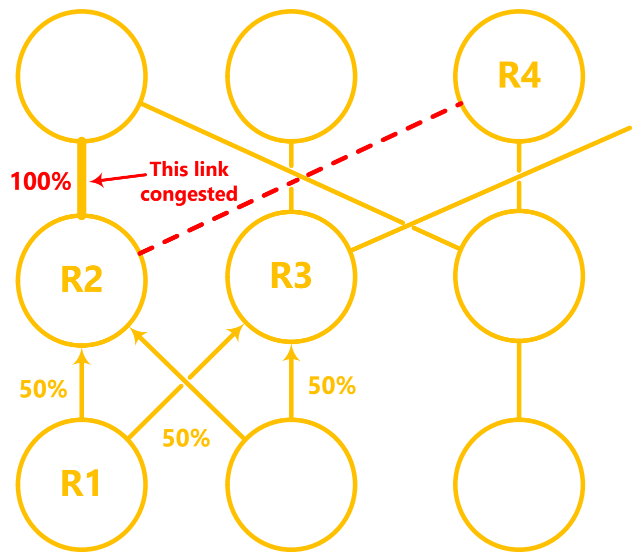
Roadmap

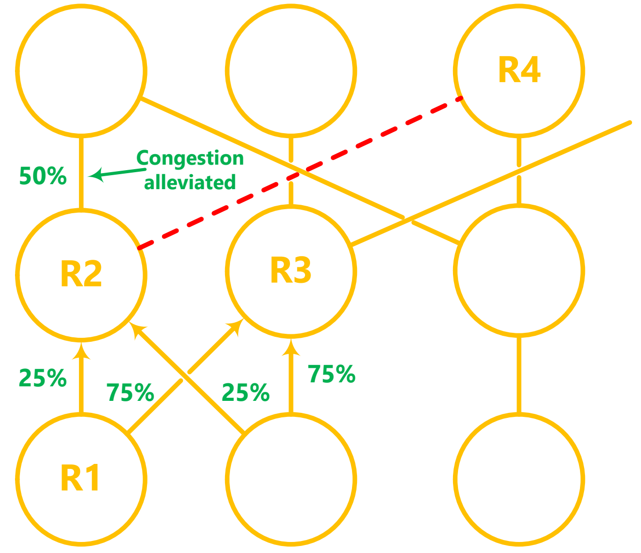
Traffic Engineering

- 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 ☹

Link b/w R2 and R4 goes down but R1 does not know that





Controller installs path with different ECMP weights

Traffic Engineering (cont.)

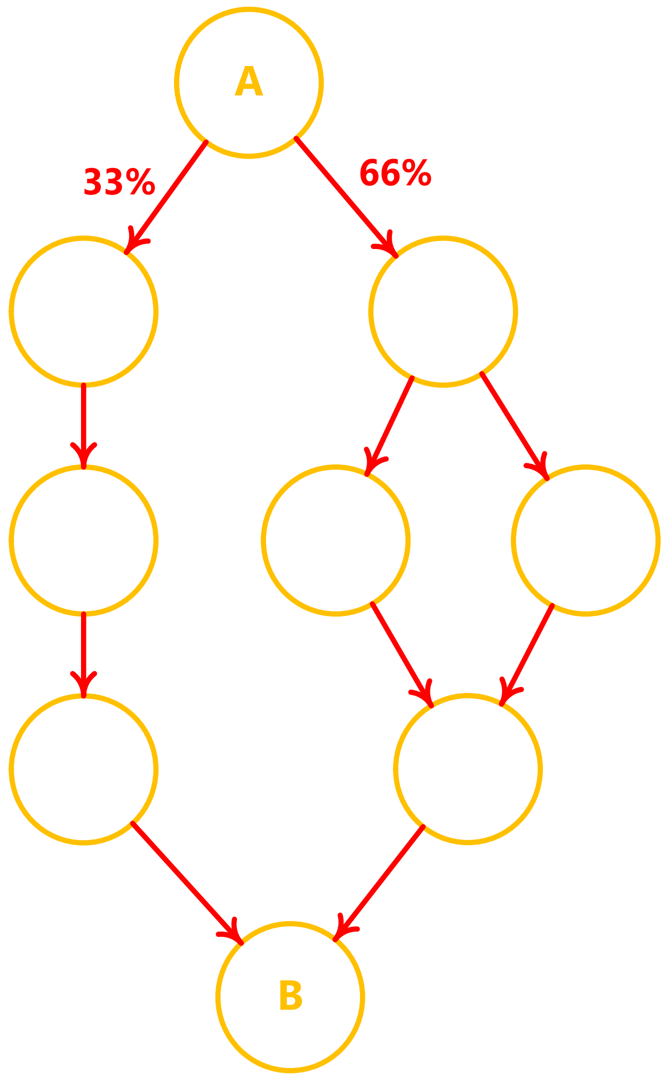
- Implementation

  Requires knowing
  - traffic matrix (TM)
  - Network topology and capacities

  Solves Linear Programming problem

  Computes 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



Ask to the vendors!

- Weighted ECMP in DC switches
  - Most common HW platforms **can do it** (e.g. Broadcom)
  - Signaling via BGP does not look complicated either
  - *Note: Has implications on hardware resource usage*

- Consistent Hashing
  - Goes well with weighted ECMP
  - Well defined in RFC 2992

- Add-Path: Receive Only
  - Not a standard (sigh)
  - We really like receive-only functionality

Conclusions

What we learned

- BGP SDN is Practical
  - Does not require new firmware or API’s
  - Though some BGP extensions are nice to have
- BGP SDN is Simple and Stable
  - BGP Code is pretty mature (for most vendors)
  - Easy to roll-back to regular routing
- BGP SDN is Efficient
  - Solves our current problems and in future allows solving much more
- TE is possible without MPLS
  - Solves our current problems and in future allows solving much more

Questions?

Contacts:

Edet Nkposong - edetn@microsoft.com
Tim LaBerge - Tim.LaBerge@microsoft.com
Naoki Kitajima - naokikit@microsoft.com

References

- BGP Routing for Data-Centers

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

- ExaBGP

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

- BGP Link-Bandwidth

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

Goals and Non-Goals of project

Goals

- Deploy on existing networks, without major 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

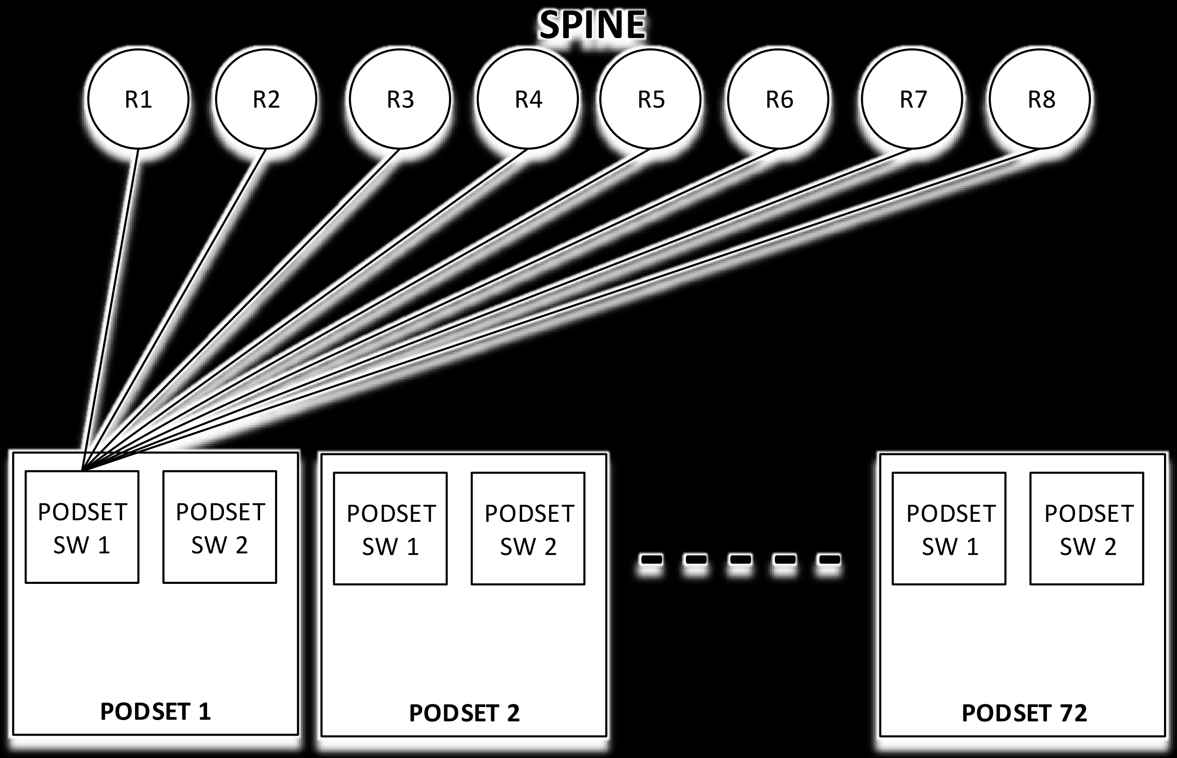
Non-Goals

- Network Virtualization or segmentation
- Per-flow, 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

Simulation Tests --- IGP vs BGP Protocol Selection

OSPF – Route Surge Test

- Test bed that emulates 72 PODSETs
- Each PODSET comprises 2 switches
- Objective – study system and route table behavior when control plane is operating in a state that mimics production



Test Bed

- 4 Spine switches
- 144 VRFs created on a router – each VRF = 1x podset switch
  - Each VRF has 8 logical interfaces (2 to each spine)
  - This emulates the 8-way required by the podset switch
- 3 physical podset switches
- Each podset carries 6 server-side IP Subnets

Test Bed

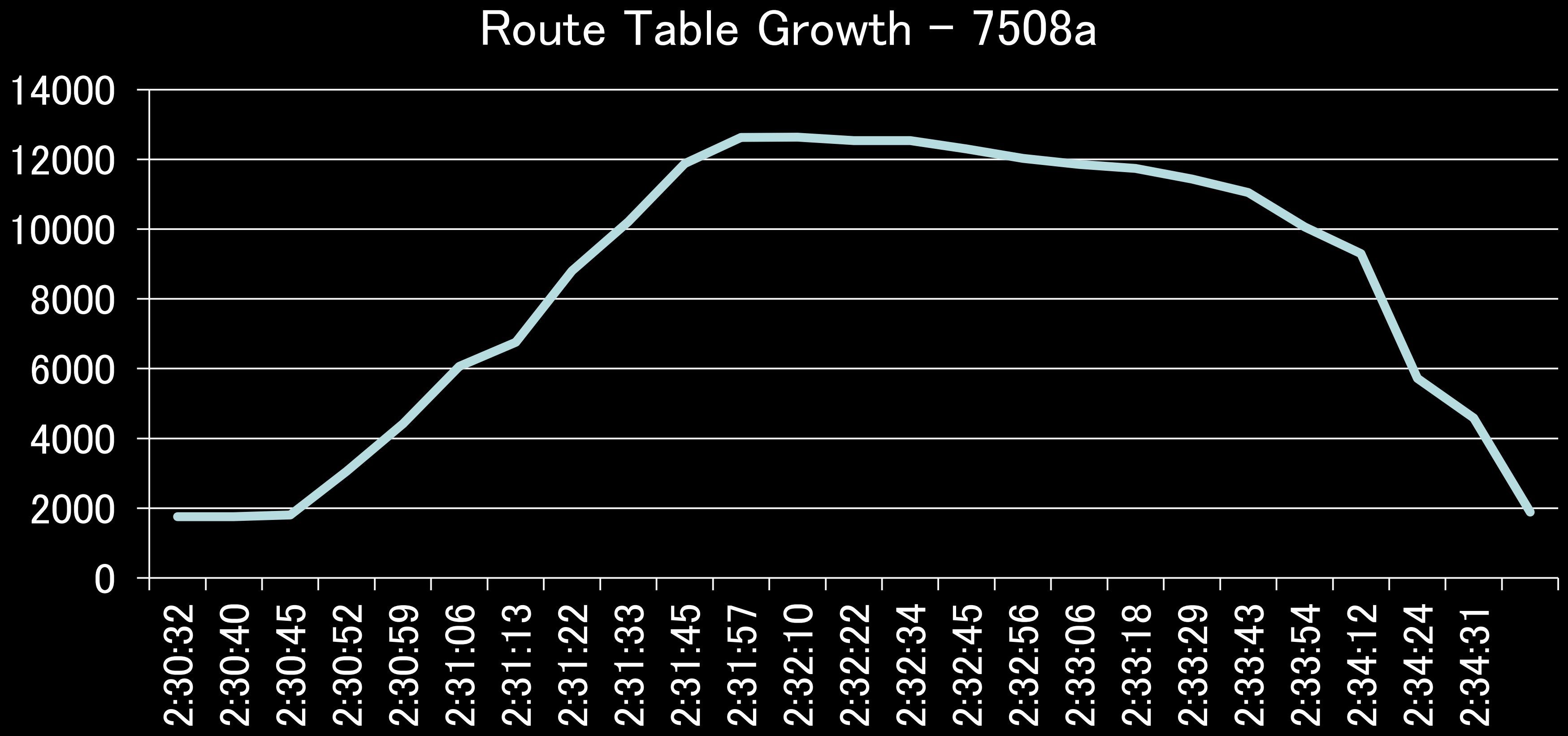
- Route table calculations
  - Expected OSPF state
    - 144 x 2 x 4 = 1152 links for infrastructure
    - 144 x 6 = 864 server routes (although these will be 4-way since we have brought everything into 4 spines (instead of 8)
    - Some loopback addresses and routes from the real podset switches
    - We expect ~ (144 x 2 x 4) + (144 x 6) – 144 = 1872 routes

- Initial testing proved that the platform can sustain this scale (control and forwarding plane) – document name

- What happens when we shake things up ?

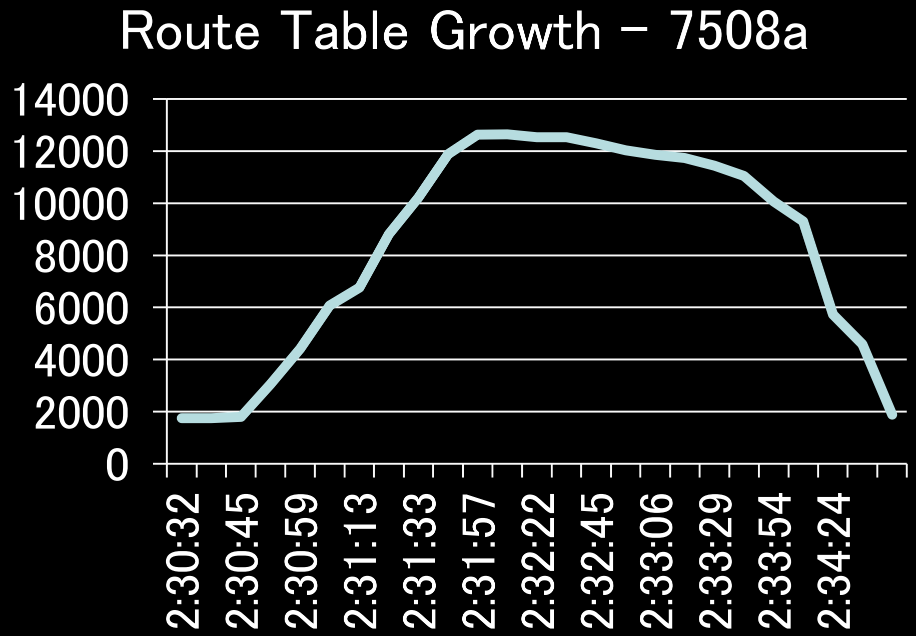
OSPF Surge Test

- Effect of bringing up 72 podset (144 OSPF neighbors) all at once

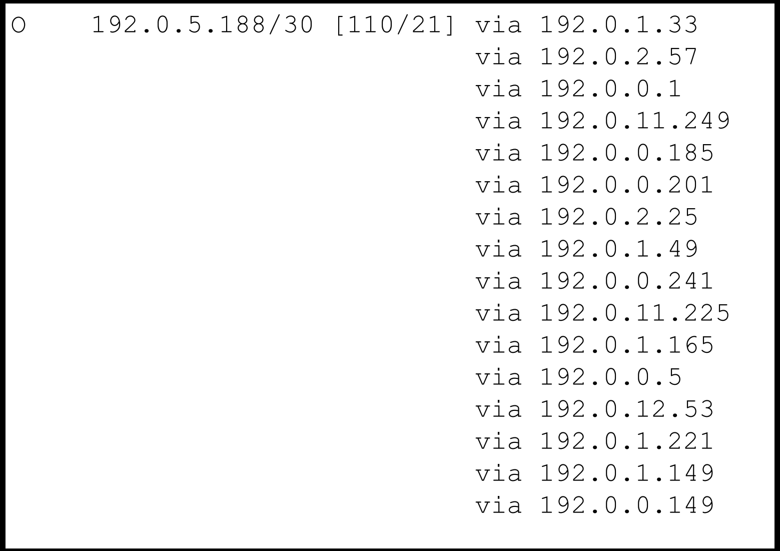


OSPF Surge Test

- Why the surge ?
  - As adjacencies come up, the spine learns about routes through other podset switches
  - Given that we have 144 podset switches, we expect to see 144-way routes although only 16-way routes are accepted



- Sample route



```
O    192.0.5.188/30 [110/21] via 192.0.1.33
                             via 192.0.2.57
                             via 192.0.0.1
                             via 192.0.11.249
                             via 192.0.0.185
                             via 192.0.0.201
                             via 192.0.2.25
                             via 192.0.1.49
                             via 192.0.0.241
                             via 192.0.11.225
                             via 192.0.1.165
                             via 192.0.0.5
                             via 192.0.12.53
                             via 192.0.1.221
                             via 192.0.1.149
                             via 192.0.0.149
```

- Route table reveals that we can have 16-way routes for any destination including infrastructure routes
- This is highly undesirable but completely expected and normal

OSPF Surge Test

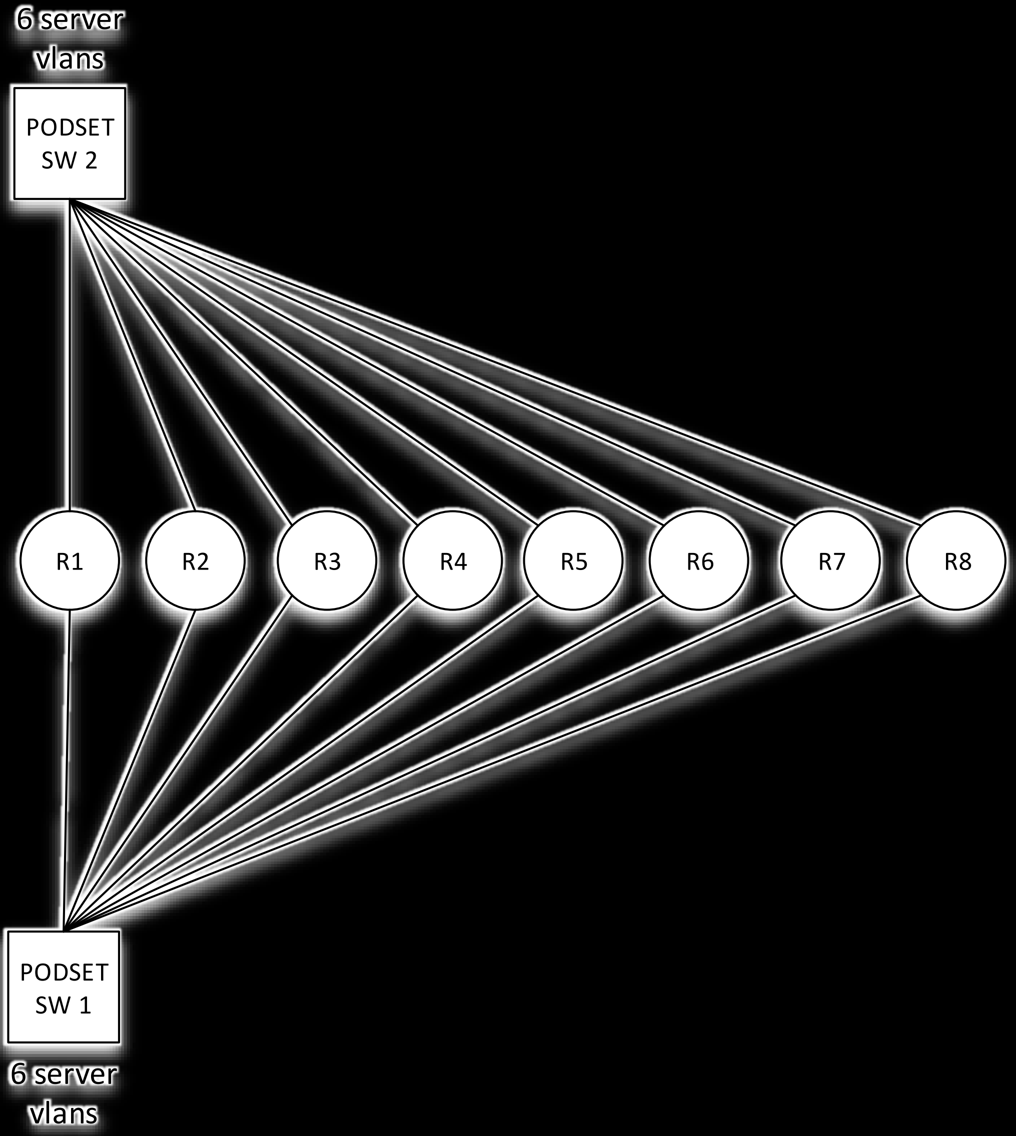
- Instead of installing a 2-way towards the podset switch, the spine ends-up installing a 16-way for podset switches that are disconnected

- If a podset switch-spine link is disabled, the spine will learn about this particular podset switches IP subnets via other shims
  - Unnecessary 16-way routes

- For every disabled podset switch-spine link, the spine will install a 16-way route through other podset switches

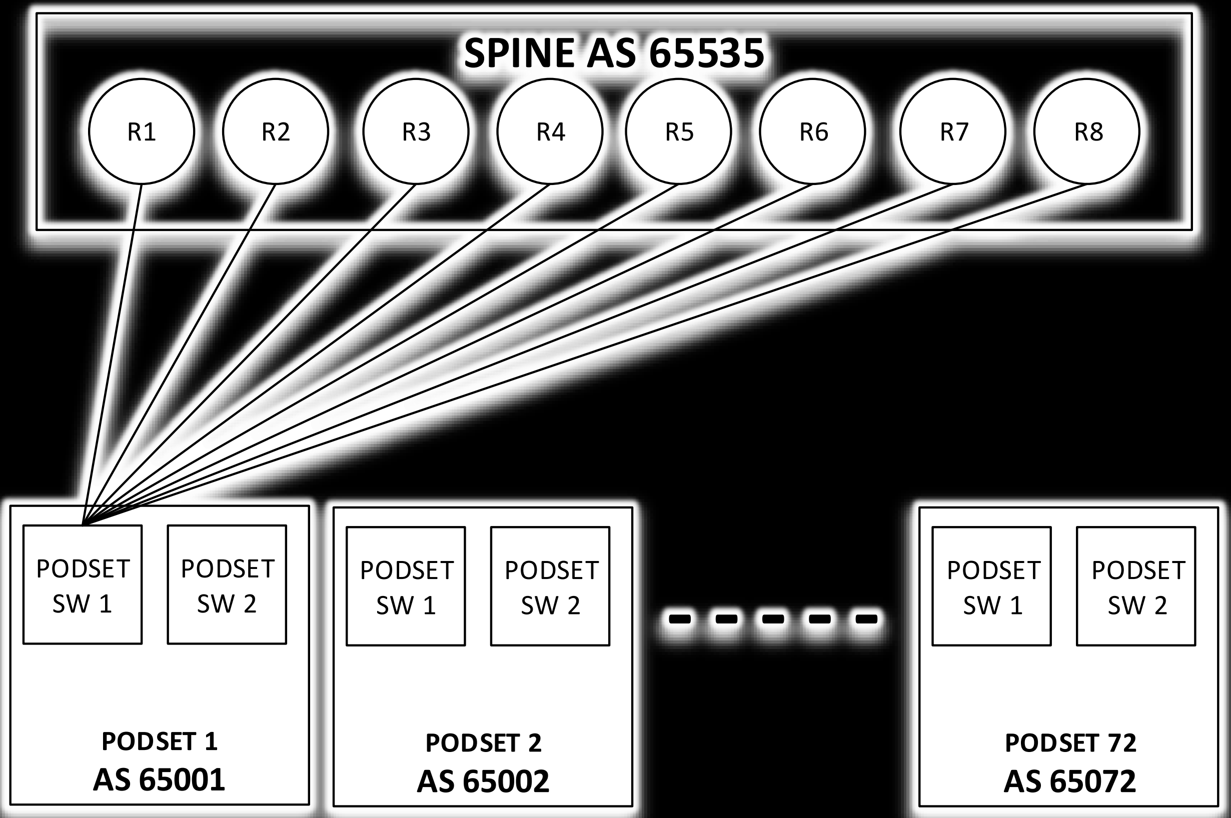
- The surge was enough to fill the FIB (same timeline as graph on slide 12)

```
2011-02-16T02:33:32.160872+00:00 sat-a75ag-poc-1a SandCell: %SAND-3-
ROUTING_OVERFLOW: Software is unable to fit all the routes in hardware
due to lack of fec entries. All routed traffic is being dropped.
```



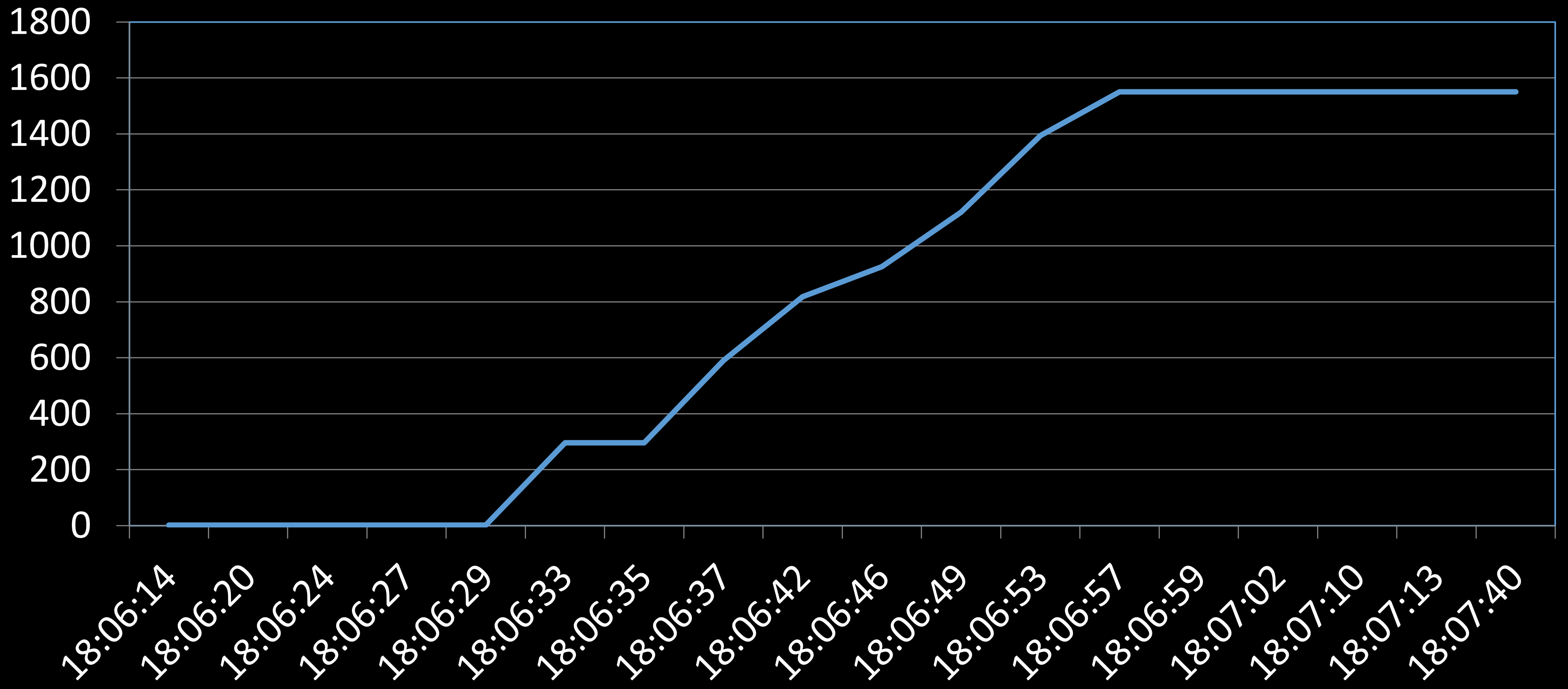
BGP Surge Test

- BGP design
  - Spine AS 65535
  - PODSET AS starting at 65001, 65002 etc



BGP Surge Test

- Effect of bringing up 72 PODSETs (144 BGP neighbors) all at once



OSPF vs BGP Surge Test – Summary

- With the proposed design, OSPF exposed a potential surge issue (commodity switches have smaller TCAM limits) – could be solved by specific vendor tweaks – non standard.

- Network needs to be able to handle the surge and any additional 16-way routes due to disconnected spine-podset switch links
  - Protocol enhancements required
  - Prevent infrastructure routes from appearing as 16-way.

- BGP advantages
  - Very deterministic behavior
  - Protocol design takes care of eliminating the surge effect (i.e. spine won't learn routes with its own AS)
  - ECMP supported and routes are labeled by the container they came from (AS #) – beautiful !