Centralium: A Hybrid Route-Planning Framework for Large-Scale Data Center Network Migrations

Yikai Lin, Mohab Gawish, Shih-Hao Tseng, Lixin Gao†, Cen Zhao, John Tracey, Sunyi Shao, Hyojeong Kim, Ying Zhang
Meta Platforms, Inc. †University of Massachusetts Amherst

Abstract

Meta's data center networks have relied on BGP for routing and interconnectivity due to its scalability and simplicity. However, as our network evolves, BGP's limitations in supporting complex network migrations have become apparent. These migrations require customized routing at each intermediate step, considering topological properties. In this paper, we highlight the unique challenges of production migration and illustrate how native BGP falls short, unable to encode both sequential and spatial conditions. To address this challenge, we introduce a novel Route Planning Abstraction (RPA) that augments the BGP protocol to support migration. It enables centralized route planning while maintaining distributed enforcement. We demonstrate the power of this abstraction by developing Centralium, a hybrid route-planning framework with a centralized controller, and over ten use cases to support various migrations in production. Our production experience with Centralium, deployed alongside BGP in large-scale data centers, has shown substantial reduction in the time and risk of network migration operations.

CCS Concepts

• **Networks** → **Data center networks**; **Routing protocols**; **Network design and planning algorithms**; **Network manageability**; **Programmable networks**.

Keywords

Centralized Routing, Distributed Routing, Network Migration, Route Planning, BGP

ACM Reference Format:
Yikai Lin, Mohab Gawish, Shih-Hao Tseng, Lixin Gao, Cen Zhao, John Tracey, Sunyi Shao, Hyojeong Kim, Ying Zhang. 2025. Centralium: A Hybrid Route-Planning Framework for Large-Scale Data Center Network Migrations. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3718958.3750519

1 Introduction

Meta's data center networks (DCNs) support the infrastructure that delivers compute and storage capabilities to satisfy the needs of billions of users. In order to accommodate the ever-increasing needs of users and applications, data center networks typically have to perform frequent network migrations. These migrations are triggered for various reasons, such as introducing new topologies, upgrading to new hardware and technology, decommissioning old equipment, expanding capacity, evolving routing design, and supporting service-specific requirements. Network migration tasks typically involve hundreds to thousands of switches, leading to significant changes in network topology and capacity. These migrations can span from days to months, requiring careful coordination among multiple teams to ensure no degradation in the performance of the production network. Naively draining an entire region and performing migration is too disruptive and results in significant loss of capacity. Furthermore, dependencies may prevent services from being completely moved out of a region, making *live network migration* a critical requirement.

A critical component in a network migration is *route planning*. A routing plan consists of a sequence of routing rules that indicate which forwarding paths to select for each destination and how to split traffic among these paths. Unlike prescribing only the paths for the current network, a route plan for a migration designs a sequence of routing snapshots for intermediate steps in the migration. Its intent can often be described in a conditional term: "If the network is in this condition, use this specific path assignment." We argue that route planning for migration is a critical but yet previously overlooked problem in the literature. In this paper, we first use production case studies and quantification to introduce the problem and stress the challenge of solving this problem.

Border Gateway Protocol (BGP) being a viable routing solution in data center networks has been well known in the industry. Thanks to its scalability and flexible policy control, BGP is used heavily throughout our production networks. In the past, we have shared production experiences on running BGP at scale in our data centers [1, 24]. However, throughout years of operational experiences, we have increasingly realized one fundamental limitation of BGP: as a purely distributed routing protocol, its inability to provide tight control during network dynamics and migrations. We elaborate on the following challenges that have been overlooked in previous research.

First, DCN migration can cause *first/last router collapse*. The CLOS topology typical of DCNs features multiple paths to reach any destination. All next-hops with equal path form an ECMP group. However, migration changes ECMP membership greatly. Sometimes a new router is inserted to the topology, creating a shorter path than all existing ones, breaking the symmetry property of DCN. All traffic will be attracted to this newly added router, a.k.a., first router. Similarly, sometimes when removing a router layer, the ECMP group size decreases. The last router left in the group attracts all traffic, creating congestion.

SIGCOMM '25, Coimbra, Portugal

ACM ISBN 979-8-4007-1524-2/2025/09
https://doi.org/10.1145/3718958.3750519

Second, supporting such network-wide migration often requires custom BGP routing policies, which can be achieved only by modifying low-level BGP attributes, such as editing the AS-path. The interdependent manipulation of BGP attributes throughout the migration process, enacted by routing policies on each node, needs to be deliberately planned to yield the desired outcome during and after the migration.

Third, BGP does not have the ability to specify conditions or encode sequential steps. BGP can only make instantaneous decisions at a given time given all disseminated updates. For example, it follows strict rules to select routes based on the available routing attributes, *e.g.,* local preference, and AS-path. During migration, operators may need to implement a different preference to accommodate the changing nature of the network.

Fourth, while it is known that fully distributed protocols can cause routing loops and black-holes during convergence, we identify a new challenge. Specifically, transient states encountered during convergence can exhaust forwarding resources. In response to organic failure and planned maintenance events, BGP goes through an active convergence process before reaching a steady state. During this process, each node potentially traverses multiple intermediate phases, each of which corresponds to a distinct, ephemeral forwarding state. The rapid succession of intermediate phases can momentarily exhaust switches' on-chip memory in which forwarding state is stored.

We contend these challenges arise from inherent limitations in fully distributed routing systems. In BGP, which was originally designed for interdomain routing where a global network view is impractical, route planning is handled locally at each routing element. In contrast, data center networks can leverage a global view to enhance route planning for migration more effectively.

Our key idea is a novel Route Planning Abstraction (RPA), achieved by integrating a RIB (Routing Information Base) policy module into the BGP daemon. It enables *explicit and direct prescription of routing intent through a simple yet effective interface*. For instance, instead of indirectly influencing BGP's route decisions through attribute manipulation, our approach allows for the installation of specific routing rules, such as *"utilize both shortest and non-shortest paths equally"* for a given destination. This ensures a fixed number of paths are used, even when shortest paths are much limited during intermediate migration steps, thereby preventing congestion.

In this paper, we introduce Centralium, a routing system that combines centralized planning with distributed enforcement. Unlike existing centralized control approaches, Centralium minimizes the role of the controller to complex route planning during migration, while relying on traditional BGP for steady-state routing decisions. The key innovation lies in the abstraction of route planning for migration using the RPA API and the corresponding modifications to BGP for distributed enforcement. More specifically, this paper presents three technical contributions.

First, we present a comprehensive analysis of the operational challenges encountered when relying solely on BGP for routing during DCN migrations. Our contributions include a migration taxonomy, real-world examples illustrating complex migration cases, and quantitative insights into the frequency of these occurrences, based on production data. To our knowledge, this is the first study to highlight the dynamic state challenges of DCN migrations, offering valuable insights for future routing research.

Second, we introduce *Route Planning Abstraction (RPA)*, a fundamental component of the Centralium platform. The RPA comprises a centralized plan and a set of simple APIs, enabling support for a wide range of use cases during migration. We present the system design and sequencing deployment methodology, which ensures safety and correctness.

Third, we demonstrate the capabilities of Centralium through over ten applications tailored to various migration scenarios. At its core, Centralium supersedes BGP's native path selection with a priority-based algorithm, allowing operators to define explicit routing rules using a list of path sets. It enables precise control over traffic distribution across multiple paths, effectively mitigating traffic funneling during migration.

Centralium has been successfully deployed in Meta's data centers for four years, managing hundreds of thousands of switches. Our operational experience with the system has significantly reduced migration time (*e.g.,* from weeks to hours) and improved network resiliency (*e.g.,* enabling half of previously disruptive backbone maintenance).

2 Background

We provide an overview of our data center network topology and overall routing design. A more-detailed introduction is presented in Sections A.1 and A.2.

Data Center Topology. Our data center network consists of five horizontal layers from bottom to top: *Rack Switches (RSWs)*, *Fabric Switches (FSWs)*, *Spine Switches (SSWs)*, *Fabric Aggregate Downlink Units (FADUs)*, and *Fabric Aggregate Uplink Units (FAUUs)*. The FAUUs connect to the backbone devices (*e.g.,* EBs) that interconnect data centers. Logical groupings of switches, such as *pod*, *plane*, and *grid*, are introduced as units of deployment. See Figure 1.

Data Center Routing. We employ two types of distributed routing protocols in our data center networks: a path-vector protocol, BGP [1], for *production prefixes*, and an in-house link-state protocol, Open/R [11], for *infrastructure prefixes* that facilitate network device connectivity, management, and diagnostics. Both protocols run concurrently on every layer of our DC network. Centralium leverages Open/R as a resilient, out of band management network while it augments BGP with its centralized intent.

Traffic Distribution. One distinct property of data center CLOS topology is the large number of parallel paths between any pair of servers. Traffic distribution is typically achieved by Equal Cost Multi-Path (ECMP) in BGP. Network faults and maintenance can cause topology asymmetry. This can negatively impact traffic distribution uniformity otherwise achieved by ECMP. We therefore employ Unequal Cost Multi-Path (UCMP) or Weighted Cost Multi-Path (WCMP) [22]. WCMP is supported by BGP by incorporating available path capacity in the hashing decision.

3 Challenges in DCN Migration

We run BGP [1] as the routing protocol for production prefixes in our data center networks for its high reliability and scalability. However, as we evolve our data center networks toward higher capacity, efficiency, and reliability, one apparent limitation to a

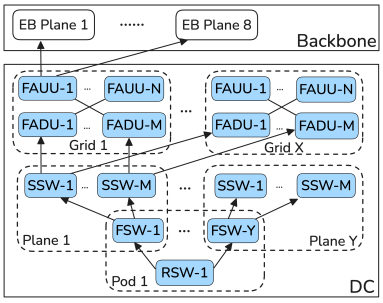


Figure 1: Abstract Meta Data Center Topology

purely distributed routing protocol is that it cannot support *route planning for network migration that lasts days to weeks*. In the following section, we explain this problem which is overlooked by the existing literature. We quantify our production data to demonstrate its significance and operational challenges.

3.1 DCN Migrations

Network migration involves modifying the network infrastructure by adding, removing, or swapping switches and circuits at the physical layer, as well as making configuration changes that may potentially disrupt traffic. This is a common operation in data center networks and can be initiated for various reasons, such as adopting new hardware and technology, decommissioning old equipment, expanding capacity, and introducing new topology or routing.

A network migration task typically affects tens to thousands of switches, often requiring physical deployment work on-site that can last for weeks. Simply draining (*i.e.,* steering traffic away from) the entire cluster or region to perform migration is not feasible, as this brute-force method would result in a significant loss of capacity, impacting service performance. Additionally, it is sometimes impossible to drain the entire region due to strict requirements on service location. Therefore, *live network migration* is necessary. The paper provides a characterization of various migrations in Table 1 and offers examples in subsequent sections.

Routing System Evolution. Our DC routing layout has evolved over multiple iterations to adapt to new application requirements, simplify topology, and eliminate legacy features. The same intent can be implemented in different ways. For example, in the past few years, we gradually updated the entire fleet from manually crafted policy to automatically compiled policy [24].

Incremental Capacity Scaling. The DC physical topology is constantly evolving to increase capacity, phase out legacy platforms, incorporate new hardware, or overhaul infrastructure to improve cooling, space, or power efficiency. It ranges from upgrading port speed to wiring a new topology [39]. It is one of the most large-scale migrations which can span over 6 months. Section 3.2 gives one such example.

Differential Traffic Distribution. Traffic is selectively allocated to specific forwarding paths based on service type. For example, we apply a special policy to anycast load-bearing prefixes for routing stability during maintenance that breaks network symmetry.

Routing Policy Transitions. Routing policy intent changes to adapt to evolving services traffic forwarding objectives. For example, some services may require conditional primary and backup policies, while others may demand custom proximity-based forwarding preferences.

Traffic Drain for Maintenance. Traffic flows are shifted on and off parts of the network during maintenance.

Table 1: Network Migration Categories

Migration	Operation Frequency	Change Scope	Typical Duration
(a) Routing System Evolution	10+/year	Multi-DC	~1.5 months
(b) Incremental Capacity Scaling	10+/year	Multi-DC	~6 months
(c) Differential Traffic Distribution	10+/year	Sub-DC	~2 months
(d) Routing Policy Transitions	10+/year	Multi-DC	~3 months
(e) Traffic Drain For Maintenance	Daily	Multi-DC	<1 hour

3.2 Scenario 1: First Router Problem in Topology Expansion

Figure 2 shows a topology composed of five layers[1]. The topology is slated for a capacity expansion by replacing two layers (FAv1 and Edge) in the initial state with a single layer with bigger capacity (FAv2) in the final state.

To perform the operation incrementally without traffic disruption, FAv2 nodes must be introduced into the traffic forwarding path before FAv1 and Edge are removed, as shown in Figure 2. The intermediate state where the two old layers and the new one coexist forms a transitory topology. Given the asynchronous nature of BGP path convergence, native BGP may result in a *first-router problem, where the first SSW to advertise the new path in the transitory topology attracts all traffic.* This is because the new path has a shorter AS-path length, which is preferred by default BGP path selection. Furthermore, given that new nodes are deliberately deployed into the production environment incrementally, not all FAv2 nodes are activated at once. The first FAv2 node to be activated will unduly attract all SSW and Backbone traffic, even after BGP converges to a steady state.

Naive approach: To avoid the funneling problem in traditional BGP, one could rely on ad-hoc policy to manipulate the BGP attributes, *e.g.,* by padding the AS-path of the route received from FAv2 by one ASN on an SSW. However, such transitory policies must be cleaned up after the migration is complete. Worse, redacting them may result in another first-router capacity collapse. For example, removing the AS-path padding on SSWs can cause FSWs to momentarily funnel northbound traffic to the first converging SSW, as it will advertise paths with a shorter AS-path length.

Production Quantification. The migration described in this example is not an exception but rather typical of day-to-day DC operations. Figure 3 shows five major categories of migrations and the average number of switches involved per layer. Figure 3 clearly indicates the migration scale is large and involves more switches at lower layers. Except for *Maintenance Traffic Drain* which typically involves hundreds of switches, most migrations involve tens of thousands of devices. Handling migrations at such scale and frequency with traditional BGP practices is prohibitively complex and costly.

[1]This example is derived from migrations on an older topology, hence the deviation from Figure 1.

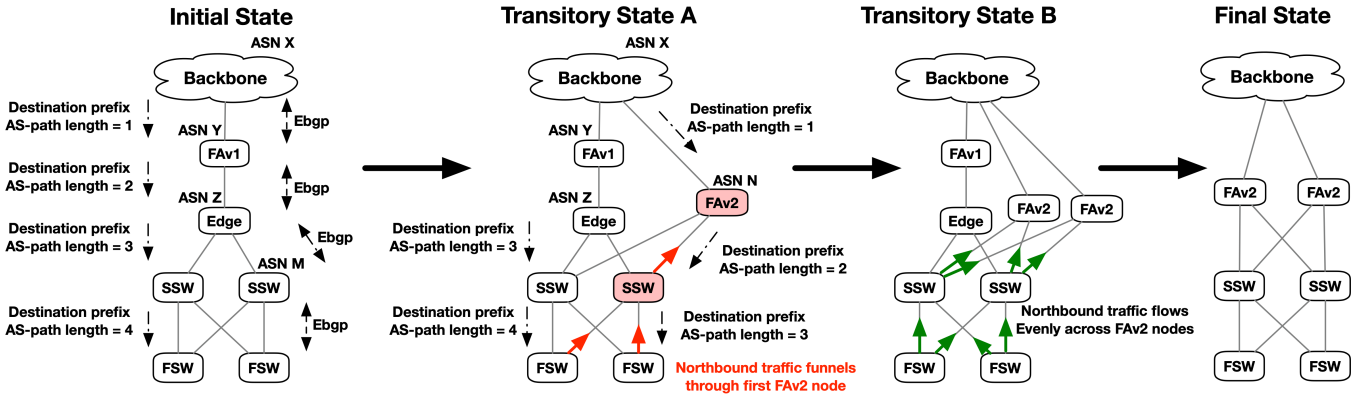


Figure 2: A topology evolution example: introducing a single switch layer to replace two existing layers. Some transitory state (*e.g.,* state A) could cause funneling of traffic due to first router problem.

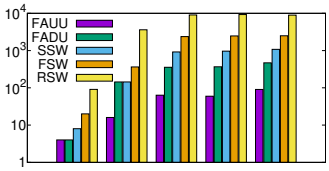


Figure 3: Average number of switches involved per layer. Left to right: Traffic Drain for Maintenance, Differential Traffic Distribution, Incremental Capacity Scaling, Routing System Evolution, Routing Policy Transitions.

3.3 Scenario 2: Last Router Problem in Decommission

We show a concrete example of the last router problem in another topology evolution scenario. Figure 4 shows a simplified topology with only two layers. The connections between SSWs and FADUs are set up such that *SSW-N* in every *plane* is connected only to *FADU-N* in every *grid* and vice versa. The objective of the migration is to decommission all SSWs and FADUs with the same numbering, say 1, to make space for new switches and additional cabling. These switches must be drained and then decommissioned.

Operators could drain all *FADU-1*s first, at which point *SSW-1*s would have zero next-hops and be safe to remove, or vice versa. Due to the asynchronous nature of drain and BGP convergence, there will be transitory states during the operation where some *FADU-1*s or *SSW-1*s are still live and attracting traffic. As shown in Figure 4, the few remaining live switches will suffer funneling of traffic (last router problem) until they get drained and the network converges to a steady state. Packets will be dropped during this time.

Ideally, switches whose number of available next-hops drops below a safe threshold should stop attracting traffic. In standard BGP this threshold is implicitly zero as BGP will only withdraw a route when there is no available next-hop.

Naive approach: Some vendors support specification of minimum ECMP [34] that allows BGP to withdraw a route when the number of available next-hops falls below a non-zero threshold. However, setting and resetting this policy configuration add additional steps and substantial delays to the migration. Enacting this policy on a selected set of switches also introduces overhead for managing these transient policy exceptions.

3.4 Scenario 3: Forwarding State Exhaustion by Intermediate Migration States

As BGP converges, prefixes are exchanged across multiple switch layers, creating transitory forwarding states. The potentially large set of forwarding states can exhaust on-chip memory resources.

Figure 5 illustrates one such example we encountered in production. EB[1:8] originate and advertise the same set of N prefixes to UU[1:4], which are then advertised downstream to DU through eight BGP sessions (*i.e.,* two sessions per UU-DU pair).

During EB[1:2] maintenance, a preset BGP export policy is applied on them such that the prefixes they advertise have less favorable attributes compared to EB[3:8], effectively transitioning EB[1:2] from *LIVE* state to *MAINTENANCE*. Given that every individual BGP session converges to the final state independently, *LIVE* EBs could assume one of the following groups during convergence: EB[1:8] (initial state), EB[2:8] (EB1 not live), EB[1, 3:8] (EB2 not live), or EB[3:8] (final, converged state). As a result, UU[1:4] may have up to four next-hop group objects for the N prefixes during convergence. A next-hop group object is a collection of next-hops to which packets that belong to the same forwarding equivalence class (FEC) are hashed. Each of the N prefixes received from EB[1:8] by each UU map to one of the four next-hop groups. The maximum number of possible transitory next-hop groups on UU[1:4] is s^m, where s is the number of states each EB switch can assume, and m is the number of EB switches changing their state. In this specific example, s and m are both 2.

Depending on the next-hop group, a given prefix is mapped to on UU[1:4], a WCMP weight is assigned, and relayed downstream to the DUs through a BGP update message in the form of a link-bandwidth community. Because each DU has eight BGP sessions toward UU[1:4], the maximum number of possible next-hop groups on each DU is up to 4^8 (65536), which far exceeds the maximum number of next-hop group objects supported by the DU hardware. The overflow of entries prevents timely update of switches' forwarding information, which could lead to packet loss.

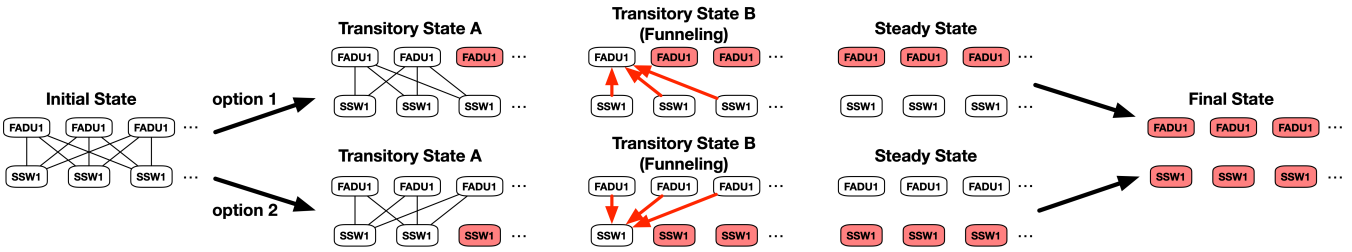


Figure 4: When decommissioning a group of interconnected SSWs and FADUs to make space for new switches and additional cabling, some transitory state (*e.g.,* state B) could cause funneling of traffic due to last router problem.

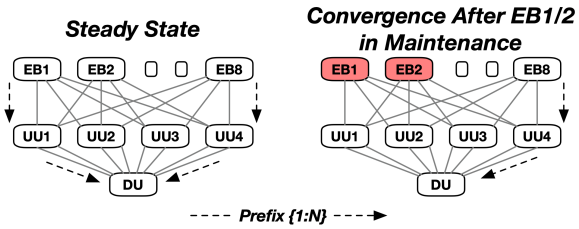


Figure 5: Transient State Explosion during Distributed WCMP Convergence.

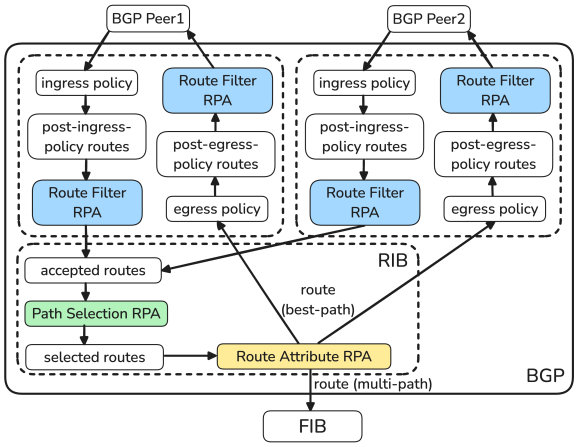


Figure 6: RPAs influence rather than take over BGP's decision making process without changing how information is exchanged between BGP peers.

Native approach: While some control plane implementations apply heuristics-based optimizations to reduce the possibility of combinatorial next-hop groups explosion [28], these attempts are best effort and are not guaranteed to provide protections in every convergence event.

4 Route Planning Abstraction (RPA)

4.1 Design Choices

While a centralized routing controller can address the problems described in Section 3 by directly prescribing routing entries on every switch, it comes with its own challenges and limitations [8]. These include, but are not limited to, the high performance requirement due to decoupling of the control plane from the forwarding elements (switches receive forwarding information from a remote controller rather than local, directly connected peers), and the trade-off between blast radius and scalability (few logical entities managing hundreds or thousands of switches).

More importantly, from an operational standpoint, it is infeasible to transition from a fully distributed protocol deployment ([1]) to a fully centralized controller solution without going through intermediate stages with incremental changes. One possible route is through vertical slicing, where fully centralized and fully distributed control-planes coexist, targeting disjoint or overlapping sets of switches ([5, 14, 30, 37]).

Based on our experience and success running BGP at scale, we decide to explore a different route where we preserve the autonomy of the distributed control plane while providing routing programmability to external entities. The key enabler to our solution is the Route Planning Abstractions (RPAs), a set of custom plug-and-play constructs in our BGP implementation that can influence the RIB computation process. In a fully centralized solution, a switch has limited routing capability without the controller, as its routing table is largely if not completely dictated by the controller. In our solution however, BGP switches remain independent. RPAs provide a set of *alternative* rules/strategies beyond BGP's local view to help it make more informed and optimal decisions. These rules and strategies can be derived and prescribed a priori, thus decoupling the distributed control plane from the entities deploying the RPAs. Operators can proactively and asynchronously augment switches with RPAs and rely on BGP to actively and independently handle real-time local network changes. For example, BGP can independently discover and process new viable routes by locally re-applying the pre-installed RPAs to update its forwarding entries. Two direct benefits of this approach are: (1) response to network events is fast due to locality; (2) policy (RPA) generation can be simple without the need to maintain precise forwarding entries directly.

In the following subsections, we introduce RPAs from the perspective of BGP control-plane workflow (Section 4.2), the different RPA APIs (Section 4.3), and how they apply to the motivating examples (Section 4.4).

4.2 RPA Augmented BGP Workflow

Figure 6 depicts an RPA-euipped BGP's control-plane workflow. The workflow starts with receiving routes from BGP peers. The routes need to first pass standard sanity checks, such as loop avoidance

and local ingress policies. The post-ingress-policy routes then go through the first RPA module, *Route Filter*. A *Route Filter* RPA specifies which IP subnets and within each subnet what prefix lengths are allowed. This is typically enacted at boundaries of network domains, such as between data centers and the backbone. Routes allowed by a *Route Filter* RPA are "officially" in the RIB. For each destination prefix, BGP needs to perform path selection to generate the multipath set (*i.e.,* set of equally preferred routes) for forwarding. Standard BGP path selection criteria would typically prefer highest local preference, shortest AS-path length, etc. [25]. Using a *Path Selection* RPA, however, operators can customize the path selection criteria and influence the routes selected (*e.g.,* by treating paths of varying AS-path lengths equally, see Section 3.2). The selected routes are then (if configured) assigned different weights for UCMP/WCMP. In a fully distributed UCMP setup, the weights are derived from local links or peer advertised capacity. Using a *Route Attribute* RPA, operators can directly prescribe what and how weights are assigned, fundamentally eliminating the churn described in Section 3.4. The weighted multipath set is then sent to the FIB (Forwarding Information Base) for forwarding. One best path is selected per destination prefix by BGP using tie breaking rules and advertised to BGP peers if it passes local egress policies and another layer of *Route Filter* RPA.

```
PathSelectionRpa {
  Statement-1 {
    Destination,
    PathSetList [
      PathSet-1 {
        PathSignature,
        MinNextHop,
      }, ...
    ],
    BgpNativeMinNextHop,
    KeepFibWarmIfMnhViolated,
  }, ...
]
```

(a) path selection

```
RouteAttributeRpa {
  Statement-1 {
    Destination,
    NextHopWeightList [
      NextHopWeight-1 {
        PathSignature,
        Weight,
      }, ...
    ],
    ExpirationTime,
  }, ...
]
```

(b) route attribute

```
RouteFilterRpa {
  Statement-1 {
    PeerSignature,
    EgressFilter {
      PrefixSet-1 {
        Prefix,
        MinMaskLength,
        MaxMaskLength,
      }, ...
    },
    IngressFilter {
      <same_schema_as_egress>
    }
  }, ...
]
```

(c) route filter

Figure 7: RPA primitives for different routing functions

4.3 RPA Primitives Design

Path Selection RPAs override standard BGP path selection with a priority-based selection algorithm. Operators describe path selection intent in the form of an ordered list of path sets. A path set is a group of operator-defined BGP paths toward a defined destination (*i.e.,* prefix). Each path set has a distinct signature common across all member BGP paths it encompasses. A signature is a unique combination of standard BGP transitive attributes that identifies a given path set. BGP attributes of member BGP paths in a path set may not completely overlap, as long as they all share their path set common signature.

Figure 7a shows its detailed structure. It is defined per group of destination prefixes that share the same intent.

- **Define custom PathSetList:** A priority list of path sets that BGP must select from for forwarding toward the defined destination. If none of the path sets is matched, BGP falls back to its native path selection. Whether a path set is matched is determined by the following two criteria:
  - **PathSignature:** A collection of BGP attribute match criteria that signify a group of BGP paths toward the specified destination. An attribute match criteria can be specified as a regular expression against BGP attributes, *e.g.,* "as_path_regex=^12345" matches AS_Paths starting with ASN 12345 **regardless of their lengths**, effectively equalizing paths of varying lengths of the same origin. This feature allows operators to prescribe paths explicitly without tampering with BGP attributes.
  - **MinNextHop:** To avoid traffic funneling when the ECMP group size shrinks, we can explicitly set the minimum next-hop value (*MinNextHop*) for each path set. If not satisfied, the current path set is not matched even if the signature matches.
- **Augment native BGP selection.** We can also enhance native BGP selection by enforcing minimum number of next-hops in *BgpNativeMinNextHop*. Note that this is different from the previous *MinNextHop* parameter, as it directly applies to BGP's natively selected paths. If this threshold is violated, since there is nothing to fall back to, BGP must withdraw the route from its peers. When this happens, operators can specify whether to keep the forwarding entries of this route so in-flight packets are not dropped.

As mentioned in Section 4.2, routes go through the Path Selection RPA after being accepted into the RIB. If a destination prefix has a matching Path Selection RPA statement, its routes will be matched against path sets in that statement. When a route's attributes match a path set's signature, it will be selected, regardless of how those attributes compare to other routes'. If an incoming route does not match any of the defined path sets, BGP falls back to its native path selection algorithm.

For every prefix in the RIB, the Path Selection RPA algorithm walks the priority list in order and selects the first path set that has matched enough active routes toward the prefix. All matching routes of the chosen path set are selected for forwarding and installed in the FIB, while the least preferred (see Section 5.3) route is advertised to peers.

Route Attribute RPAs capture operator's desired traffic distribution ratio among possible paths toward a destination prefix in an asynchronous fashion. Unlike in a purely centralized solution where routing table entries need to be computed concretely and timely, Route Attribute RPA can be specified a priori and enacted asynchronously when paths are observed and selected for forwarding by BGP. This is possible because of RPA's abstract design, allowing the routing table entries to be computed in a delayed fashion by the protocol control plane.

- **Traffic forwarding behavior across paths** is described in the list *NextHopWeightList* which encompasses: 1) a BGP path set which is defined by signatures of BGP attributes (*PathSignature*, same as in Path Selection RPA) and 2) an integer value *Weight* indicating the relative weight of paths matched by Signature.
- **Operation parameters.** We can also define the condition of those forwarding behaviors. One example is *ExpirationTime*, specifying when the statement will be invalidated and BGP falls back to its native traffic distribution solution (ECMP or distributed WCMP).

Route Attribute RPAs support traffic engineering solutions that directly prescribe the desired traffic distribution on every switch, fundamentally eliminating the transient state explosions we described in Section 3.4. Operators can update prescribed weights using an RPA in anticipation of upcoming maintenance, and rely on BGP control plane to update the routing entries when the devices actually go down.

Route Filter RPAs allow operators to dynamically set what prefixes can be exchanged between any BGP peers without changing the routing policy or path selection criteria.

- **Peer signature:** We define the set of BGP peers where the policy is applied (*PeerSignature*).
- **Prefix filter:** Since our origination and propagation policies are deterministic, we choose to apply an allow list for what prefixes can be exchanged between BGP peers.
- **Prefix attribute:** To avoid leaking more specific prefixes, we can set the maximum length of prefix mask. This is crucial in cases where we only expect a prefix aggregate, and incorrectly accepting too many specific prefixes can overload the compute and forwarding resources in switches.

4.4 Case Studies

In this section, we showcase how we use RPAs to resolve the topology migration scenarios described in Section 3.2 and Section 3.3.

In our BGP configuration, we typically attach a designated BGP community to all prefixes at their point of origin [1]. For example, default route (*i.e.,* 0.0.0.0/0 and ::/0) prefixes advertised downstream by the backbone have the "BACKBONE_DEFAULT_ROUTE" community attached.

4.4.1 Equalizing paths of varying AS_Path lengths. In Section 3.2, the main objective is to equalize paths of varying AS_Path lengths from the backbone in order to not cause traffic funneling during the migration. We described how a solution based on AS_Path editing in BGP routing policy can still be problematic. Using RPA, we can easily override BGP's path selection algorithm by directly prescribing our objective:

```
PathSelectionRpa {
  Statement-1 {
    Destination: "BACKBONE_DEFAULT_ROUTE",
    PathSetList [
      PathSet-1 {
        Signature: AS_Path [Backbone_ASN,...]
      }]}}
```

In the above RPA snippet, we provide two pieces of information to BGP: what destination prefixes this RPA applies to, and how to select paths toward those destinations. By simply specifying "select paths that start with the backbone AS_Path number," BGP neglects the lengths of AS_Paths and selects both old (longer) and new (shorter) paths for forwarding. With this RPA in place, the topology migration can be carried out in a non-disruptive fashion with no policy residues post-migration. The RPA can just be removed, restoring BGP to its native path selection.

4.4.2 Localized and custom capacity collapse prevention. In Section 3.3, the main objective is to prevent transient funneling caused by switches asynchronously going down, an inherent limitation of distributed routing protocols. We described how this migration is impossible to execute safely without resorting to certain minimum capacity protection configurations. As introduced in Section 4.3, Path Selection RPAs support specification of minimum next-hop size for both custom and BGP native path selection algorithms. Additionally, since this RPA is a per-switch override, we can selectively inject the following RPA in the SSWs to be decommissioned:

```
PathSelectionRpa {
  Statement-1 {
    Destination: "BACKBONE_DEFAULT_ROUTE",
    PathSetList [], # empty
    BgpNativeMinNextHop: 75%,
    KeepFibWarmIfMnhViolated: True,
  },
}
```

RPAs allow us to trivially enforce a custom minimum capacity threshold in a selected subset of devices unique to this particular migration. With the protection of this RPA, the migration is significantly simplified into two steps: draining all FADUs in selected spine planes, draining all SSWs in the selected spine planes. There is no funneling (due to minimum next-hop) and no black-holing (due to FIB kept warm) throughout the entire process.

5 System Implementation

In Section 4, we showed how RPA augments BGP's distributed decision making by providing programmability to route planning. In this section, we focus on our system implementation that makes RPAs instrumental in production. As depicted in Figure 3, migrations typically involve coordinating hundreds or thousands of switches. At this scale, it is non-trivial to generate and deploy RPAs quickly and safely, and to ensure their consistency across the fleet. To this end, we designed and deployed a logically centralized controller that facilitates BGP route planning in our large-scale DCNs. The Centralium controller provides the following critical functions:

1. **Pre-deployment network health checks.** The controller ensures the networks meet the prerequisites for the desired RPA. This can include BGP policy and binary versions, specific RIB states (as prescribed by operator), and general network health (*e.g.,* congestion freeness).
2. **Per-switch RPA generation.** The controller consumes operator's high-level intent for route planning and compiles the specific RPAs for all target switches.
3. **Coordinated RPA deployment across the fleet.** Centralium deploys the new RPAs to target switches in a phased manner following a safe order that does not disrupt live traffic.

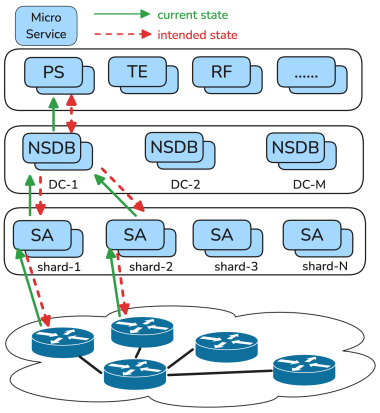


Figure 8: Centralium architecture - Centralized Controller (Apps, Network State Database, Switch Agent) over distributed BGP switches

4. **Post-deployment network health checks.** Centralium monitors the health of the network as deployment happens, and checks for expected changes to RIB and FIB (*e.g.,* new paths are selected).
5. **Consistency guarantee of desired RPAs in the fleet.** The controller continuously tracks desired RPAs on every switch and ensures all target switches (particularly those re-provisioned or newly commissioned) are up-to-date.

We detail the design and implementation of the controller in Section 5.1, discuss the scalability and reliability of it in Section 5.2, and highlight two challenges that we solved on interoperability between native and RPA-augmented BGP switches in Section 5.3.

5.1 System Architecture

The Centralium controller is logically divided into three layers: application, storage - Network State Database (NSDB), and I/O - Switch Agent (SA), as depicted in Figure 8. Each layer performs the following functions:

- Application Layer: implements business logic and generates intended network state in forms of RPA and configurations. We have onboarded 10+ use cases, including Path Selection, Traffic Engineering, and Route Filtering.
- Storage Layer: hosts the "superset" of current state (ground truth) and intended state (intent) of the network
- I/O Layer: continuously collects current network state from physical switches, and reconciles current state with intended state by augmenting distributed control-plane

Figure 8 also captures the two continuous data flows in Centralium: one that propagates current network state (*i.e.,* ground truth) northbound, and one that propagates intended state southbound. Both data flows converge and feed into the storage layer (hence the "superset"), then propagate further, on demand and often in a slimmer size, to their consumers.

Services are the units of deployment that carry out data communications between layers in the controller. One of the key design decisions we made is to enforce service uniformity through a common template [15]. Most notably, every Centralium service maintains **two contrasting network views**: an **intended state**, which captures what applications want network state to be, and a **current state**, which captures the actual network state (*i.e.,* ground truth) from the physical switches. Every service consumes or populates these two states according to its role. For example, the Switch Agent (1) consumes intended state and writes it to the distributed control-plane to reconcile current state with intended state, and (2) polls or streams state and statistics from physical switches to populate the current state.

This design decision has multiple implications.

- Providing consistency guarantee. One of the biggest challenges with managing a large-scale network is ensuring consistency across the fleet - be it policy or configuration. Contrasting network views allow Centralium to easily detect straggler switches, and provide an eventual consistency guarantee by continuously reconciling both states.
- Enabling customized rollout plans. Developers can leverage this visibility to easily implement customized rollout plans in their applications. For example, one can trivially implement a slow roll process gated by the percentage of managed devices that are out-of-sync.
- Significant code reusability. Significant portions of the service implementation can be reused. For instance, all services share the same pub/sub modules, health check module, and APIs. As a result, our command line interface tooling, dashboards, and external monitoring are compatible with all existing services.

Network State Representation. Current and intended network states in Centralium share the same tree representation, rooted at a device map. This allows any node to be identified programmatically via a path string. Coupled with Thrift [2] encapsulation, all services share the same set of generic get/set/publish/subscribe APIs that are data-agnostic. These APIs can be used externally (for RPCs) and internally: when instantiating the publisher module for example, services are actually subscribing to their local current or intended state for any changes to publish. In Section A.3, we share an example of Centralium's data model and generic API that supports wildcards.

5.2 Scalability and Reliability

RPA preserves independence of the distributed control plane which is highly scalable and reliable, and Centralium is designed to take advantage of it. Compared to a fully centralized SDN solution, where (1) routing applications demand a high fidelity and fresh representation of the network to practically replace the distributed control plane, and (2) forwarding elements depend on the centralized controller to populate and maintain their forwarding tables, Centralium is (1) scalable as it consumes more abstract and less time-sensitive network state, and (2) simple and reliable as it doesn't need to replace the entire distributed control-plane. Here, we briefly cover some additional scalability and reliability measures.

Sharding. By separating the I/O, storage, and application layers, the Centralium controller can scale horizontally. Each layer can be configured with its own number of shards and replicas per shard, based on its specific requirements. Centralium's NSDB service can easily handle thousands of switches in one data center with one

shard and two replicas. In Section 6.1, we show CPU and memory usage from production deployment. As our data centers continue to evolve, Centralium's modular and configurable design allows it to easily adapt to larger scale.

Service Failures. NSDB is the core of Centralium's data flows. Other services publish or subscribe to NSDB. This requires NSDB to have high availability and low write latency. As a result, we adopt an eventual consistency model. All publish requests are fanned out to all NSDB replicas, while read requests are directed to the elected leader. If a replica fails, read requests get automatically re-routed to the next elected leader. Other services are also configured with more than one replica with auto leader election. All replicas perform read operations, while only the leader writes to NSDB.

Device Failures. Switch Agents continuously talk to the distributed control plane to reconcile current state with intended state. If a device becomes unreachable, Centralium is capable of correctly identifying the intended device operational state, for example down for maintenance, and alerting network operators of unexpected device unavailability.

5.3 Interoperability Between RPA and Non-RPA Switches

The asynchronous nature of RPA deployment and activation means there will be (transient) co-existence of RPA and non-RPA[2] switches during each deployment. We describe two scenarios where this can lead to routing loops or funneling and how we eliminate such risks with two RPA design and deployment principles.

5.3.1 BGP Path Dissemination. The criteria by which a given BGP speaker chooses which routes to advertise to its peers is crucial, as bad decisions can cause unstable routing states and convergence problems. For instance, Figure 9 shows a topology with a mix of RPA-augmented and native BGP speakers; R6 is a BGP speaker with Path Selection RPAs, while R[1-5] run native BGP with multi-path enabled. Each peer establishes eBGP with its directly connected peers, and each switch belongs to a different Autonomous System. RPAs are configured to enable R6 to load-balance traffic destined to *Prefix D* over R2 and R5. R1 receives Prefix D from some upstream peer and advertises it downstream. R6 selects the two paths it receives from R2 and R5 for forwarding. At this point, R6 needs to select one of the two paths for advertisement. If R6 chooses the path it receives from R2 for advertisement to R5 (left figure), R5 will end up with two paths with equal AS_Path length. Given that R5 utilizes multi-path, they will both be chosen for forwarding. This decision will install a persistent routing loop between R5 and R6, as they become on each others' forwarding paths.

Given that path selection and advertisement are closely related in path-vector protocols, replacing the path selection algorithm must be accompanied by its path advertisement rules to avoid routing loops. In our implementation, we preserve loop avoidance with one addition to the standard BGP loop prevention mechanism: RPA BGP speakers must advertise the path with the least favorable BGP attributes among the ones it selects for forwarding [31, 38]. For

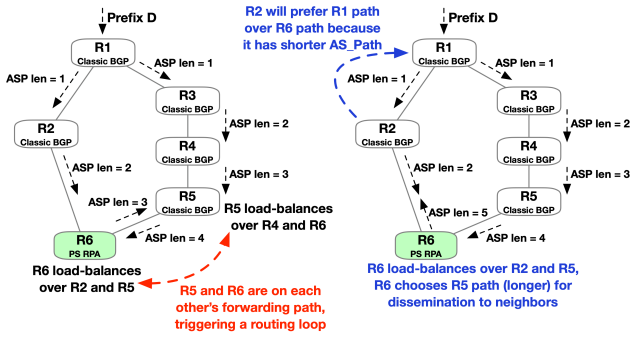


Figure 9: Bad BGP path dissemination causes routing loop (left); Disseminating highest cost (longest AS_Path) prevents routing loop (right)

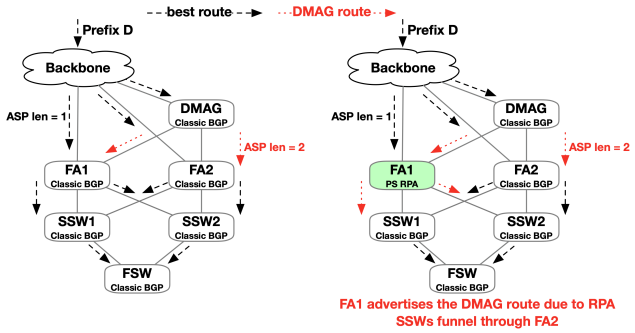


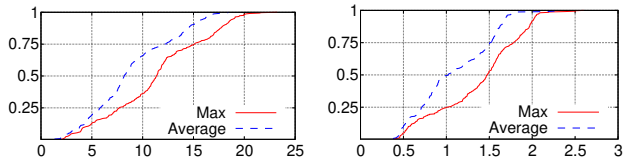
Figure 10: Uncoordinated RPA deployment can lead to transient funneling

example, an RPA BGP speaker will advertise the path with the longest AS_Path among the paths it selects for forwarding.

By applying this rule (right figure in Figure 9), R6 will no longer advertise the path it receives from R2 to R5, as it is not the one with the highest cost (longest AS_Path) among the paths it selects for forwarding. Instead, R6 will advertise the path it receives from R5, and following standard distance-vector split-horizon rule, paths selected for forwarding cannot be advertised back to the peer from which they were received. As such, R6 can advertise the path it receives from R5 only to R2, and a routing loop no longer forms.

5.3.2 RPA Deployment Sequencing. Since RPAs could influence path selection and in turn path dissemination, the order in which they are deployed can also be impactful. In Figure 10, we describe such a scenario where uncoordinated RPA deployment could lead to transient traffic funneling. In this example, the prefix D is originated by the backbone and propagated down into the DC. In the initial stage where all switches are running native BGP, the direct path between FA and the backbone is preferred over the longer backup path through DMAG. Suppose we want all DC switches to utilize both paths (*i.e.,* ignoring AS_Path length), we can deploy a Path Selection RPA similar to that described in Section 4.4.1 on FSWs, SSWs, and FAs. Suppose we do not coordinate the deployment of this RPA, and the new RPA takes effect on FA1 first. FA1 will select both paths for forwarding, and advertise the longer path through DMAG to its peers (as described in Section 5.3.1). Because none of

[2]In practice it is common to have multiple orthogonal RPAs on a switch. Orthogonal RPAs influence exclusive sets of prefixes, *i.e.,* a switch can be treated as non-RPA when reasoning about the behavior of a new RPA, as is in this discussion.



(a) CPU Utilization (%) **(b) Memory Utilization (GB)**

Figure 11: Per-Task Resource Consumption

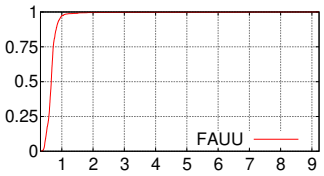


Figure 12: CDF of RPA Deployment Time (ms)

the other switches has picked up the new RPA yet, they will still prefer a shorter path. In this case SSWs will prefer the shorter path from FA2 and all northbound traffic will funnel through FA2 until FA2 gets the new RPA and advertises the longer path.

To eliminate this risk, we adopt a safe deployment practice based on knowledge of our DC topology: ordering by layer from bottom to top (*i.e.,* FSW->SSW->FA). Every layer must receive the new RPA after all their downstream peers have picked up the new RPA. This way northbound traffic is distributed over all available paths before, during, and after the migration.

This principle is summarized as: a new RPA must be deployed starting from the layer furthest from the source of the route origination; removal of an existing RPA must start from the layer closest to the source of the route origination.

6 Evaluation

In this section, we present production data of our hybrid system's performance and scalability. We also quantify the significant complexity reduction and time savings RPA enabled in different network migrations.

6.1 Scalability

We deploy Centralium controller services using our own containerization platform *Twine* [35]. Each micro-service (*i.e.,* job) is typically run with two identical replicas (*i.e.,* tasks) with leader election for redundancy. In total there are 10-20 Centralium tasks in a data center.

Centralium's most resource hungry services are NSDB and Switch Agent. In Figure 11, we show CDFs of CPU and memory usage, respectively, across all NSDB and Switch Agent tasks. In Figure 11 (a), we see that their single-core-equivalent CPU utilization peaks out below 25%, with 75% of tasks never exceeding 15%. In Figure 11 (b), we see that their memory consumption peaks out well below 3GB, with 50% of tasks never exceeding 1.5GB.

These low hardware resource consumption numbers reflect how lightweight and scalable the controller is. Thanks to RPA, it needs only abstract state, such as network topology, rather than full routing state, to augment the protocol control plane and prescribe a priori intent. RPA allows the controller to offload the most computation- and storage-heavy functions at the more scalable layer: the thousands of physical switches in each DC.

6.2 Performance

RPA generation is significantly less compute-intensive compared to generating full routing table entries as the computation is kept in the distributed protocol control plane. Per our measurement, the controller is able to consistently generate RPAs for a full DC in under 200 milliseconds.

RPA deployment can happen a priori in an asynchronous fashion thanks to its abstract design. In few cases such as Traffic Engineering, a short deployment time is still crucial for a timely routing re-convergence. In Figure 12, we show a distribution of RPA deployment time (how long it takes to update RPAs in BGP via RPC). The results are collected for the FAUU layer, as they are physically the most distant from server racks, where Centralium services are running. Most RPA updates complete within one millisecond.

RPA evaluation. When an RPA is deployed in BGP, it needs to be evaluated against all routes in the RIB as their next-hops and weights might have changed. Once evaluated, the matched RPA statement is cached so future re-evaluation on the same route is much faster. In Table 2 we show collected RPA evaluation time for both cache misses and cache hits.

6.3 Operational Efficiency

Since we deployed RPA in production, our network operators have shipped over 400 commits (150+/year) to create or update RPAs for a wide spectrum of routing use cases. For the network migrations we introduced in Section 3.1, we calculate the number of steps on the critical path (*i.e.,* steps that are strictly in-order) and the number of days (based on our average push cadence of three weeks [1]) it takes to complete each migration, with and without (Path Selection) RPA, shown in Table 3. While the specific numbers are highly dependent on our production environment (*e.g.,* number of devices, velocity and frequency of software binary and configuration updates), the drastic difference between those with and without RPA reveals its capability in simplifying and protecting disruptive network operations.

6.4 Network Efficiency

With RPA and a centralized controller, we can fundamentally eliminate the transient forwarding state explosions described in Section 3.4. Figure 13 highlights the effectiveness of our TE solution between DCN and backbone. Our TE algorithm consumes network topology and minimizes maximum link utilization to improve effective network capacity (*i.e.,* the amount of traffic that can be handled without congestion). As shown in Figure 13, our TE consistently performs close to theoretical optimum (ideal WCMP), and not-surprisingly better than ECMP. This improvement in effective capacity enabled up to 45% of maintenance events that would have otherwise been blocked due to Service Level Agreement violations.

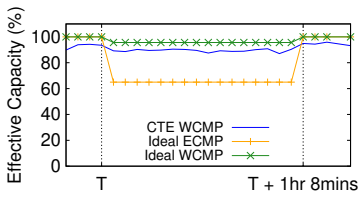


Figure 13: Near-optimal Centralized TE via RPA

Table 2: RPA evaluation time per route (ms)

	p50	p95	p99
w/o cache	<1	2	4
w/ cache	<1	<1	<1

Table 3: Table 1 extended with RPA-enabled reduction and time savings in different network migrations

	#Steps w/o RPA	#Steps w RPA	#Days w/o RPA	#Days w/ RPA	RPA LOC
(a)	2	1	42	<1	300-1000
(b)	9	3	189	21	200-300
(c)	3	1	63	7	50-100
(d)	5	3	105	21	100-200
(e)	3	1	<1	<1	<50

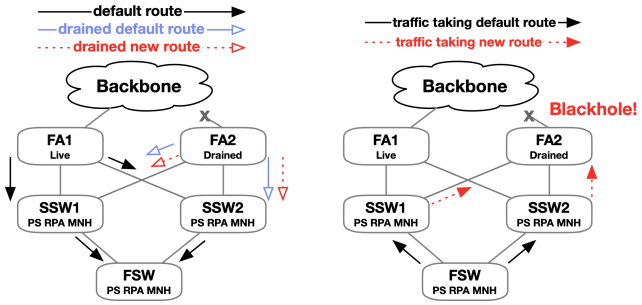


Figure 14: Major Site EVent (SEV) where incorrect Path Selection RPA adds to traffic black-holing.

7 Operational Experience

In this section, we discuss our production experience operating and evolving Centralium for four years.

7.1 Dependency and Verification

Centralium is a hybrid system that involves both functional and configuration *dependencies* between its centralized and distributed components. These dependencies, which do not exist in fully distributed solutions and are far less pronounced in fully centralized ones, introduce unique and non-trivial challenges in our daily operations. For instance, any evolution of the RPA feature requires coordinated changes to both the controller and BGP binaries, which are released through two separate pipelines operating on drastically different cadences (days versus weeks). In some cases, the typically agile centralized controller must effectively *wait* for hundreds of thousands of switches to be updated to avoid incompatibility issues. Another example is RPA's reliance on the underlying base BGP policies. As described in Section 4.3, operators can use any combination of BGP attributes to identify routes (destinations) and paths (*PathSignature*). This dependency means that RPA relies on these attributes being correctly specified by the base BGP policy, which is defined by a different system [24]. Uncoordinated deployment of RPA and base BGP policies can lead to unexpected routing behavior.

We systematically addressed these challenges through enhanced pre-deployment verification and unified routing change orchestration. The former builds upon our existing emulation test suite used for BGP binary and configuration qualification [1]. We introduced new integration tests that validate end-to-end routing intent by emulating a reduced-scale production network incorporating both BGP and the controller. These tests run whenever there is an update to the binaries or configuration, preventing incompatible changes from reaching production. The latter is a new application on top of the Centralium controller, which orchestrates the rapid and safe deployment of base BGP policies and RPA across the fleet.

7.2 Debuggability and Usability

While RPA enables migrations that are infeasible with standard BGP policies (Section 3) and generally simplifies and accelerates migrations (Section 6.3), reasoning about RPA's behavior—especially Path Selection RPA—remains non-trivial.

Defining and using RPAs correctly in migrations requires operators to have a good understanding of both the migration requirements and the implications of different RPA settings. The explicitness and flexibility of RPA, while powerful, are also prone to mis-configurations. Figure 14 shows one such example: operators want to originate a new route (more specific than default route) from FAs. A Path Selection RPA with *BgpNativeMinNextHop* was pre-deployed on SSWs and FSWs to protect this operation from causing funneling (*i.e.,* a switch should only advertise this route when sufficient next-hops are available). However, the *KeepFibWarmIfMnhViolated* knob was incorrectly set (*i.e.,* allowing the new route to be installed in the FIB despite not being advertised), a nuanced and seemingly harmless setting. During the migration, an FA that was not production ready (*i.e.,* missing cabling toward backbone) unexpectedly originated the new route, which did not propagate beyond SSWs but got installed to their FIBs due to RPA. As a result, packets reaching SSWs via the default route ended up taking the more specific new route toward the bad FA and got black-holed. If the *KeepFibWarmIfMnhViolated* knob was not set, packets would have taken the default routes toward all FAs but the drained one. It remains a challenge and research direction to capture migration intent using high-level abstractions or language and auto generate RPAs.

Reasoning About Routing Behavior with RPAs. RPAs are independently specified and deployed, often in an ad-hoc fashion, in different parts of the network. This is in contrast to base BGP policies, which are to a large extent consistent across different data centers and familiar to operators. The custom intent and scope of RPAs, often known by a few operators, make it difficult for other operators to reason about the routing behavior of an RPA-switch (*i.e.,* which route should be admitted/selected/advertised), let alone the entire network. We alleviate the single-switch problem by enhancing our debugging tooling to (1) show all active RPAs on a switch, and (2) highlight the active RPA given a particular route. However, it remains a challenge to reason about a network's holistic routing behavior when ad-hoc RPAs are applied.

7.3 BGP Configuration versus RPA

Path selection was the first and motivating routing function for RPA. We gradually introduced new functions (*i.e.,* route attribute and route filter) as we gained confidence with Centralium and discovered more use cases. An important question that keeps coming up as we evolve RPA over the years is whether a new routing function should be implemented as BGP configuration or RPA. Our decision making process boils down to the following key factors (Figure 15):

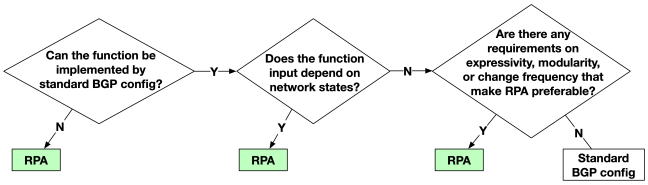


Figure 15: How to choose between BGP configuration and RPA for implementing new routing functions.

- Viability of standard BGP configuration. Migrations that require non-standard path selection (*e.g.,* Section 3.2) are examples where BGP configuration is not viable.
- Dependency on network dynamics. BGP configuration is better suited for static functions that are agnostic to real-time network topology or traffic distribution. RPA, on the other hand, can leverage the holistic view of the centralized controller for more dynamic functions, such as traffic engineering.
- Expressivity, modularity, and agility. Compared to BGP configuration, RPA is more explicit, more modular (*i.e.,* each RPA component is specified and deployed independently), and can be deployed much more timely via Centralium. These qualities also play a huge role in our decision of how to implement a new routing function.

7.4 Applicability to Other Networks

Applying to Small/Medium Networks. A key aspect of Centralium's design and deployment is ownership of the BGP daemon and the ability to modify it. While this may be a prerequisite for direct applicability, the high-level route plan can still be implemented indirectly by using an external compiler to translate it into low-level BGP primitives. However, this indirect approach is more difficult to reason about and can increase the risk of errors. On the other hand, BGP extensions remain an active area of standardization. We believe the concept of RPA could be integrated into open-source BGP daemons and potentially advanced through standardization efforts to benefit the broader community.

Applying to Other Networks at Meta. We are actively exploring the use cases of Centralium at Meta beyond data centers. We have identified routing requirements for AI backend networks [9], where the network evolves at an even faster pace and multiple versions of topology coexist. Centralium can also provide finer-grained routing control considering training job placement. Additionally, we have been using Centralium to control how traffic flows into the backbone network, facilitating backbone network migrations.

Applying to Other Routing Protocols. The design and implementation of RPA are heavily influenced by Meta's BGP deployment, where each node lacks a holistic view of the network. However, we believe the core concept of RPA—achieving hybrid routing control through an externally programmable component within the protocol control plane—can be extended to other routing protocols such as OSPF and Open/R.

8 Related Work

Software Defined Networking (SDN) is a groundbreaking innovation in the networking community in the past decade [20]. It not only stimulates new research [3, 10, 32] but also lands in large-scale deployed systems [8, 12, 23, 29]. At Meta, although we have chosen the traditional BGP due to our own requirements and challenges [1], we continue exploring a hybrid approach for the benefits of both sides [24, 26]. The closest to ours are Google's Jupiter [29] and Orion [8] which are data center SDN solutions that directly instrument the data plane. While our choice of BGP provides fault tolerance, our work addresses its unique transient challenges.

Intent-Based Routing. Expressing routing objectives in a more human-readable language has drawn significant interests in recent years. Jinjing [36] introduces LAI to express ACL update synthesis. Propane [3] introduces RIR to express constraints on policy. Propane/AT [4] and SyNet [7] use their own Domain Specific Language or existing techniques to express intents. These efforts focus more on policy specification and synthesis, while Centralium emphasizes how to use policies to control routing in a centralized system. Centralium leverages our high-level routing intent platform [24] and injects policy at the RIB level.

Network Management. In the broader network configuration space, network management systems such as Robotron [33] and MALT [21] use high-level intents to low-level device configurations with minimal human intervention. Our work focuses on higher level routing intent which can be translated to Robotron-like systems.

Dynamic Network Updates. There have been several efforts on dynamic network updates in the SDN context [13, 17–19]. zUpdate [16] and Snowcap [27] determine a transition plan from one configuration to another. CrystalNet provides an emulation platform to test changes before production. They address different challenges avoiding packet drops due to inconsistent rules [18].

9 Conclusion

Centralium tackles route planning problems for complex DCN migrations. It enhances BGP by introducing centralized control through Route Planning Abstractions (RPAs), which explicitly and directly prescribe intent. This hybrid approach combines centralized route planning with distributed route enforcement, addressing the limitations of traditional distributed routing protocols and the challenges associated with full centralization. Our production experience demonstrates the effectiveness of this approach in overcoming the limits of traditional BGP. The integration of centralized and distributed routing remains an active area of research, driven by the evolving demands of modern, large-scale data centers.

Ethics: This work does not raise any ethical issues.

Acknowledgments

This work is a close collaboration within Meta's Network Infrastructure team, especially between Network Routing, Data-center Network Engineering, FBOSS, WAN Systems, and Core Networking teams. We would like to thank Omar Baldonado, our shepherd Peter Steenkiste, and the anonymous reviewers for their constructive and valuable feedback on earlier drafts of this publication.

References

[1] Anubhavnidhi Abhashkumar, Kausik Subramanian, Alexey Andreyev, Hyojeong Kim, Nanda Kishore Salem, Jingyi Yang, Petr Lapukhov, Aditya Akella, and Hongyi Zeng. 2021. Running BGP in Data Centers at Scale. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX

Association, 65–81. https://www.usenix.org/conference/nsdi21/presentation/abhashkumar
[2] Apache. [n. d.]. Thrift. https://thrift.apache.org. Accessed: 2024-01-09.
[3] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2016. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 328–341.
[4] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2017. Network Configuration Synthesis with Abstract Topologies. *SIGPLAN Not.* 52, 6 (June 2017), 437–451. https://doi.org/10.1145/3140587.3062367
[5] Marcel Caria, Admela Jukan, and Marco Hoffmann. 2016. SDN partitioning: A centralized control plane for distributed routing protocols. *IEEE Transactions on Network and Service Management* 13, 3 (2016), 381–393.
[6] Marek Denis, Yuanjun Yao, Ashley Hatch, Qin Zhang, Chiun Lin Lim, Shuqiang Zhang, Kyle Sugrue, Henry Kwok, Mikel Jimenez Fernandez, Petr Lapukhov, et al. 2023. Ebb: Reliable and evolvable express backbone network in meta. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 346–359.
[7] Ahmed El-Hassany, Petar Tsankov, Laurent Vanbever, and Martin Vechev. 2017. Network-wide configuration synthesis. In *International Conference on Computer Aided Verification*. Springer, 261–281.
[8] Andrew D. Ferguson, Steve Gribble, Chi-Yao Hong, Charles Killian, Waqar Mohsin, Henrik Muehe, Joon Ong, Leon Poutievski, Arjun Singh, Lorenzo Vicisano, Richard Alimi, Shawn Shuoshuo Chen, Mike Conley, Subhasree Mandal, Karthik Nagaraj, Kondapa Naidu Bollineni, Amr Sabaa, Shidong Zhang, Min Zhu, and Amin Vahdat. 2021. Orion: Google's Software-Defined Networking Control Plane. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 83–98. https://www.usenix.org/conference/nsdi21/presentation/ferguson
[9] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, et al. 2024. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 57–70.
[10] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 58–72. https://doi.org/10.1145/2934872.2934891
[11] Saif Hasan, Petr Lapukhov, Anuj Madan, and Omar Baldonado. 2017. Open/R: Open Routing for Modern Networks. https://engineering.fb.com/2017/11/15/connectivity/open-r-open-routing-for-modern-networks/. Accessed on Jan 26, 2024.
[12] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. 15–26.
[13] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. 2014. Dynamic scheduling of network updates. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 539–550.
[14] Sajad Khorsandroo, Adrián Gallego Sánchez, Ali Saman Tosun, José M Arco, and Roberto Doriguzzi-Corin. 2021. Hybrid SDN evolution: A comprehensive survey of the state-of-the-art. *Computer Networks* 192 (2021), 107981.
[15] Yikai Lin. 2020. *Enabling and Improving Centralized Control in Network and Cyber-Physical Systems: An Application-Driven Approach*. Ph. D. Dissertation. University of Michigan, Ann Arbor.
[16] Hongqiang Harry Liu, Xin Wu, Ming Zhang, Lihua Yuan, Roger Wattenhofer, and David Maltz. 2013. zUpdate: Updating data center networks with zero loss. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. 411–422.
[17] Hongqiang Harry Liu, Yibo Zhu, Jitu Padhye, Jiaxin Cao, Sri Tallapragada, Nuno P. Lopes, Andrey Rybalchenko, Guohan Lu, and Lihua Yuan. 2017. CrystalNet: Faithfully Emulating Large Production Networks. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 599–613.
[18] Ratul Mahajan and Roger Wattenhofer. 2013. On consistent updates in software defined networks. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. 1–7.
[19] Jedidiah McClurg, Hossein Hojjat, Pavol Černỳ, and Nate Foster. 2015. Efficient synthesis of network updates. *Acm Sigplan Notices* 50, 6 (2015), 196–207.
[20] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review* 38, 2 (2008), 69–74.
[21] Jeffrey C. Mogul, Drago Goricanec, Martin Pool, Anees Shaikh, Douglas Turk, Bikash Koley, and Xiaoxue Zhao. 2020. Experiences with Modeling Network Topologies at Multiple Levels of Abstraction. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 403–418. https://www.usenix.org/conference/nsdi20/presentation/mogul
[22] Prodosh Mohapatra and Rex Fernando. 2018. *BGP Link Bandwidth Extended Community*. Internet-Draft draft-ietf-idr-link-bandwidth-07. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-idr-link-bandwidth/07/ Work in Progress.
[23] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, Rishi Kapoor, Stephen Kratzer, Nanfang Li, Hong Liu, Karthik Nagaraj, Jason Ornstein, Samir Sawhney, Ryohei Urata, Lorenzo Vicisano, Kevin Yasumura, Shidong Zhang, Junlan Zhou, and Amin Vahdat. 2022. Jupiter Evolving: Transforming Google's Datacenter Network via Optical Circuit Switches and Software-Defined Networking. In *Proceedings of ACM SIGCOMM 2022*.
[24] Sivaramakrishnan Ramanathan, Ying Zhang, Mohab Gawish, Yogesh Mundada, Zhaodong Wang, Sangki Yun, Eric Lippert, Walid Taha, Minlan Yu, and Jelena Mirkovic. 2023. Practical Intent-driven Routing Configuration Synthesis. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 629–644. https://www.usenix.org/conference/nsdi23/presentation/ramanathan
[25] Yakov Rekhter, Susan Hares, and Tony Li. 2006. A Border Gateway Protocol 4 (BGP-4). RFC 4271. https://doi.org/10.17487/RFC4271
[26] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. 2017. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 418–431.
[27] Tibor Schneider, Rüdiger Birkner, and Laurent Vanbever. 2021. Snowcap: synthesizing network-wide configuration updates. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 33–49.
[28] Vipul Shah. [n. d.]. BGP In-place Adjacency Replace (IAR). https://www.arista.com/en/support/toi/eos-4-23-2f/14449-bgp-in-place-adjacency-replace-iar. Accessed: 2020-03-13.
[29] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM computer communication review* 45, 4 (2015), 183–197.
[30] Yash Sinha, K Haribabu, et al. 2017. A survey: Hybrid sdn. *Journal of Network and Computer Applications* 100 (2017), 35–55.
[31] J.J. Garcia-Luna-Aceves Srinivas Vutukury. 1999. A Simple Approximation to Minimum-Delay Routing. In *ACM SIGCOMM Computer Communication Review, Volume 29, Issue 4*. 227–238.
[32] Peng Sun, Ratul Mahajan, Jennifer Rexford, Lihua Yuan, Ming Zhang, and Ahsan Arefin. 2014. A network-state management service. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 563–574.
[33] Yu-Wei Eric Sung, Xiaozheng Tie, Starsky HY Wong, and Hongyi Zeng. 2016. Robotron: Top-down network management at facebook scale. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 426–439.
[34] Himanshu Tambakuwala and Sanoop Ranjan. [n. d.]. BGP Minimum ECMP. https://community.juniper.net/blogs/himanshu-tambakuwala/2024/07/25/bgp-minimum-ecmp. Accessed: 2025-01-16.
[35] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, et al. 2020. Twine: A unified cluster management system for shared infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 787–803.
[36] Bingchuan Tian, Xinyi Zhang, Ennan Zhai, Hongqiang Harry Liu, Qiaobo Ye, Chunsheng Wang, Xin Wu, Zhiming Ji, Yihong Sang, Ming Zhang, Da Yu, Chen Tian, Haitao Zheng, and Ben Y. Zhao. 2019. Safely and automatically updating in-network ACL configurations with intent language. In *Proceedings of the ACM Special Interest Group on Data Communication* (Beijing, China) *(SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 214–226. https://doi.org/10.1145/3341302.3342088
[37] Shih-Hao Tseng, Ao Tang, Gagan L Choudhury, and Simon Tse. 2019. Routing stability in hybrid software-defined networks. *IEEE/ACM Transactions On Networking* 27, 2 (2019), 790–804.
[38] Iljitsch Van Beijnum, Jon Crowcroft, Francisco Valera, and Marcelo Bagnulo. 2009. Loop-freeness in multipath BGP through propagating the longest path. In *2009 IEEE International Conference on Communications Workshops*. IEEE, 1–6.
[39] Yihao Zhao, Xiaoxiang Zhang, Hang Zhu, Ying Zhang, Zhaodong Wang, Yuandong Tian, Alex Nikulkov, Joao Ferreira, Xuanzhe Liu, and Xin Jin. 2023. Klotski: Efficient and Safe Network Migration of Large Production Datacenters. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 783–797.

A Appendix

Appendices are supporting material that has not been peer-reviewed.

A.1 Data Center Topology

Figure 1 illustrates Meta's DC topology. All equipment within a given rack is connected via a single *Rack Switch (RSW)*. Each RSW connects to a group of *Fabric Switches (FSWs)*, each of which connects to a group of *Spine Switches (SSWs)*. These three layers comprise a *fabric network*. Within the fabric network, switches map to logical groupings, such as *pod* and *plane*. A pod is the smallest unit of deployment in a DC consisting of a group of interconnected FSWs and RSWs. A plane consists of a group of interconnected SSWs and FSWs.

Multiple fabrics are aggregated by a *Fabric Aggregate (FA)* layer. This layer plays a pivotal role in managing traffic flow across DCs, *i.e.*, east/west traffic, as well as traffic ingress to and egress from DCs. The FA architecture consists of a group of switches. Fabric Aggregate Downlink Units (FADUs) face down, toward DCs. Fabric Aggregate Uplink Units (FAUUs) face up toward the wide-area network. FAUUs connect to the backbone, the network that orchestrates the global interconnection of data centers [6]. FAUUs and FADUs are grouped into *grid*s. Every SSW connects to one FADU in every grid.

A.2 Data Center Routing

Since our previous work [1], we evolved our data center routing design significantly. First, we continue to use Border Gateway Protocol (BGP) to route *production prefixes*, leveraging BGP's rich policy control at every hop. Production prefixes carry high-volume application traffic and must have optimal routing with sufficient capacity at every hop in all partial failure scenarios. Second, we employ an in-house link-state protocol, Open/R[11], to route *infrastructure prefixes*, leveraging Open/R's link-state flooding and SPF-based routing paradigm. Infrastructure prefixes facilitate network device connectivity, management, and diagnostics. They carry relatively low traffic. In the event of a device or link failure, their reachability can be supported by non-shortest paths. Third, both BGP and Open/R run concurrently on every layer of the DC network. Fourth, Centralium incrementally takes on more routing functions provided by BGP. Centrailum's intended end state is the authority over path selection, traffic engineering, and traffic distribution optimization.

The new DC routing design reduces risk associated with centralized control. Such control can be exercised with minimal risk of circular dependencies or losing reachability to network devices, as it controls only BGP and accesses network devices via routes provided by Open/R. In a sense, Open/R acts as a resilient out of band (OoB) management network with in-band network properties, offering diverse accessibility paths between Centralium and each DC network node.

Within Meta's network infrastructure, BGP and Open/R policies are configured to govern the sharing of routing information and to exercise control over traffic flow objectives. These objectives encompass aspects like traffic load-balancing, redundancy, and path preference. The configuration of BGP and Open/R policies is maintained homogeneously across network tiers [24].

A.3 Centralium Data Model and APIs

Network State Representation:

```
"devices": {
  "device-1": {
    "basic": {
      "name": "device-1",
      "role": "fauu",
      ...
    },
    "bgp": {
      "route_planning_abstraction": {
        "path_selection": {...},
        "route_attribute": {...},
        "route_filter": {...}
      }
    },
  ...
}
```

Generic Path-Based API:

```
binary GetState(
  GetStateRequest {
    isIntended: false, # current state
    paths: [
      {"raw": "devices"},
      {"any": true}, # wildcarding
      {"raw": "bgp"},
      {"raw": "route_planning_abstraction"},
      {"raw": "path_selection"},
      {"raw": "statements"},
      {"regex": "vip.*"}, # regex
      ...
    ]
  }
)
```