# Delegating Data Plane with Cloud-Assisted Routing

Prasun Kanti Dey, *Member, IEEE,* and Murat Yuksel, *Senior Member, IEEE*

***Abstract*—Data centers are embracing the software-defined networking (SDN) as it is evident that this technology of completely separating the data plane from the control plane gives more flexibility for their internal routing management and provides better Quality-of-Service (QoS) to the users. Implementing a similar solution to serve the purpose of software-defined wide-area networking using public Internet routing is also gaining popularity. Although, instead of having a complete separation, a hybrid approach to keep *most* of the control plane along with the *least* of the data plane in the remote and vice-versa for the local platform may be more fitting. To this end, we propose a new hybrid SDN approach, Cloud-Assisted Routing (CAR), that utilizes the high computational services that cloud offers at a lower price by splitting both control and data plane functions between a local router and a remote cloud computing platform. Such delegation of data plane to a third-party authority requires proper control plane management policies and handling or avoiding possible loops and failures. We detail the architectural view of CAR, address its associated challenges, and present prototype-based evaluations of it for reducing routing table sizes.**

***Index Terms*—Routing scalability, Software-defined networking, Cloud-assisted networking**

## I. Introduction

THE never-ending development of newer protocols and the widespread usage of wireless communication, multi-homing, and other associated technologies pose new challenges for the existing Internet architecture. The Internet continues to witness the backbone router's Forwarding Information Base (FIB) growth faster than the router hardware technology development [4]. Along with the traditional factors like multi-homing, load balancing, and address fragmentation, new challenges like the ongoing transition to IPv6 and rapid expansion of the Internet of Things (IoT) devices contribute to the explosion of the routing tables [5]. Satisfying the demanding performance requirements as well as maintaining simplicity and automation in network management have led to the need for more flexible and granular traffic management. The *best-effort* delivery nature of the Internet is not an option anymore. Without a resilient and fault-tolerant design which is capable of rendering fine-grained control and management operation [6], existing architecture will not be adequate to offer Internet usage as a communication commodity.

P. K. Dey (e-mail: pdey@mathworks.com) is with The MathWorks Inc. He was with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, USA during most of this work. M. Yuksel (e-mail: murat.yuksel@ucf.edu) is with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL, USA.

Initial studies leading to this manuscript were published in conferences [1], [2]. The baseline concept of Cloud-Assisted Routing was patented by the U.S. Patents and Technology Office [3].

This work was sponsored in part by U.S. National Science Foundation awards 1814086 and 1647189.

Algorithmic optimizations and extensive ingestion of Application-Specific Integration Circuit (ASIC) memories are merely enough to tackle the routing scalability issue as routing has grown beyond just selecting the shortest end-to-end path. Internet Service Providers (ISPs) are expected to carefully deal with policy compliance issues, properly manage unexpected route flaps resulting from private networks' sudden connectivity or discontinuity, and immediately adapt to link or routing device failures [4]. Such dynamic changes require continuous monitoring and immediate route update propagation throughout the network for maintaining the Internet's consistency and reliability. Further, the complexity of maintaining proprietary network devices continues to grow as these purpose-built devices become more feature-rich.

SDN has proven to be an attractive solution which decouples the data plane from the control plane and supports the independent development of these two planes by allowing on-the-fly fixes and patches. In particular, SDN architecture advocates moving the control plane activities into a user-programmable software, making the network management process flexible and vendor-agnostic. With the development of open interfaces like OpenFlow [7], offloading control plane decisions to a separate, or even remote, controller for centralized optimization has already gained its popularity. In parallel, cloud computing has become the de-facto option for on-demand virtual services, an example of centralization of functions. As people are willing to access third-party computing resources (for storage and other software services) over the Internet, cloud providers are in a position where, by mixing-and-matching existing services and resources, they can come up with a unique and cost-effective hybrid design that involves virtualizing network (hardware) components. Further, they can also commercialize low-level network functions like routing to the ISPs.

To address the alarming increase in the inter-domain routing complexity, we introduce a new Internet routing architecture, *Cloud-Assisted Routing* (CAR). Figure 1 illustrates a birds-eye view of routing scalability and flexibility trade-offs that lead to the CAR approach. Considering the fact that complete separation between the control and data plane may not offer the best performance when it comes to the scalability and flexibility [8], the figure shows why the proposed hybrid approach is more likely to better tackle the routing challenges. For example, earlier works proved NP-hardness of the load balancing problem in existing SDN due to switch's constrained TCAM (Ternary Content-Addressable Memory) [9], and highlighted the lack of focus on dynamic adaptation of the number of controllers and their placement in a distributed SDN control plane (instead of statically provisioned) to react on sudden traffic pattern change [10]. By leveraging the cloud's high computational capability and cheap memory resources, CAR will perform these complex routing functions more

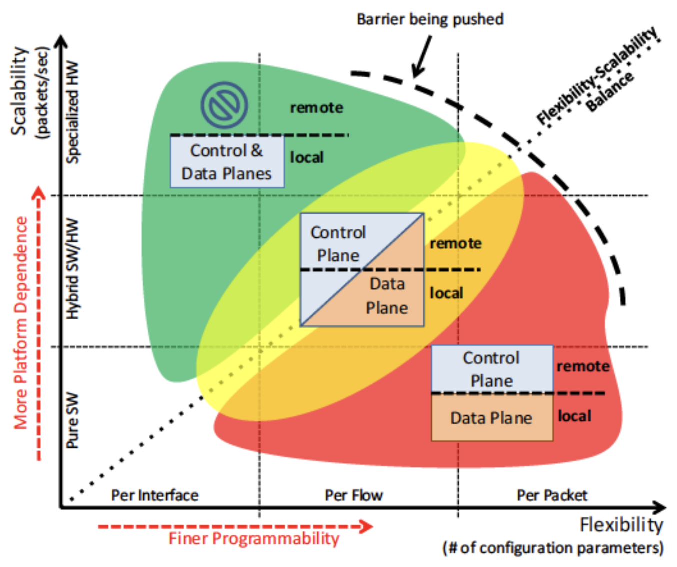


Fig. 1: Routing scalability and flexibility trade-offs [2]: Traditional routing architecture (green) places both the control and data planes in the local router, legacy SDN (red) places the control plane to a remote platform, and CAR (yellow) advocates for a hybrid SDN approach with the least of the data plane and the most of the control plane functions delegated to a remote platform.

conveniently.

CAR advocates for a hybrid approach that maintains most of the data plane and least of the control plane tasks at the router level while delegating least of the data plane and most of the control plane to a remote platform. Such a hybrid solution that maintains the high priority tasks at the physical router, locally, and offloads *partial* responsibilities to a remote platform, like cloud, to employ an adaptive cloud-router integration framework will come handy. We shall detail the functionality and responsibility separation later in Sections III-A and III-C. In general, heavyweight and larger time-scale control plane tasks such as computing Spanning Tree and generating and maintaining full-fledged IP routing tables should be the first ones to offload to a remote platform; however, lightweight and urgent responsibilities like forwarding data packets should be the first ones to be performed at the local router.

In this paper, we discuss the critical architectural opportunities and challenges that are associated with hybrid integration of cloud and physical router. In particular, we focus on the opportunistic placement of data plane functions to a remote cloud platform and propose frameworks for failure-handling and attaining loop-freeness when data packets are forwarded to a cloud platform as part of their end-to-end paths. Major contributions of our work include:

- Outlining a framework on how to integrate cloud computing resources with the physical routers and defining the operational regions where such integration is beneficial,
- Detailed analysis of data plane delegation and its benefits,
- Identifying possible loop scenarios and proposing two suggestive ways to avoid them, and
- Developing and deploying CAR prototype in a test-bed to show the proof-of-concept.

In our earlier work, we presented CAR's economic benefits [11] and initial architectural concepts [1]. This paper extends these studies by *a)* detailing how to design a loop-free data plane delegation, *b)* suggesting resolutions for failure-response incidents and achieving failure-resiliency, and *c)* detailing the CAR prototype development process.

The remainder of the paper is organized as follows. In Section II, we review the related works and extend our motivation behind this work. Section III details CAR architecture and its function placement principles. It also introduces the challenges of hybrid SDN model supported by the cloud to gain more scalability and robustness. Section IV articulates the possible failure handling and potential benefits of cloud integration in routing services, continued by the discussion on loop scenarios for data plane delegation and suggest the potential avoidance mechanisms in Section V. Next, Section VI elaborates our Emulab [12] based initial prototype of CAR, while Section VII extends our proof-of-concept effort by implementing the CAR setup on GENI [13] testbed and evaluates the FIB table size reduction. To conclude, Section VIII summarizes our observations and discusses possible future work.

## II. Motivation and Related Work

SDN technology has been the envoy of high flexibility by offering programmability and the loose coupling between the control and data planes. SDN's basic premise is to ensure an exquisite adaptation of the system's functionality without modifying much of the existing underlying infrastructure. In a typical OpenFlow-supported SDN architecture, each new flow needs to consult with the controller to decide its outgoing port. This delay, resulting from the flow going up to the controller and coming down to the switch, adds up and causes extra latency. Handling this consistently growing delay becomes challenging and earlier works have pointed out this issue [14]. Recent controllers are considerably more capable (can support up to $10^5$ requests/sec) compared to previous versions (NOX could sustain only 30K requests/sec [15]). However, it is questionable whether these efforts, which primarily focused on the datacenter-centric issues, are enough to support the Internet's data-extensive applications.

Moving the management and control plane tasks to the cloud has picked up pace [16], [17] mainly due to the elegant flexibility in network administration, even though it may be at the price of degrading the performance and the risks of exposing critical routing services to potential cascading failures [18] and attacks [19]. Early works (e.g., CARPO [20] and DISCO [21]) explored a correlation-aware flow- or switch-based traffic consolidation algorithm for placing correlated flows on the same network to maximize the overall utilization. Assuncao et al. [22] also suggested redirecting network traffic to an alternate path which is enough for serving the total traffic. Although these efforts mainly focused on switch's energy efficiency, they significantly improved the data-center network delay performance at the same time. Furthermore, with the increasing availability of 5G, Multi-Access Edge Computing (MEC) and edge cloud adoption rates are accelerating. The 5G design specification enables MEC to have complete support for user and application mobility via *local routing*, *user plane selection*, and *supporting local area data network* among other fundamentals [23] of 5G networks. Latency-sensitive and highly interactive applications such as Virtual/Augmented Reality, e-Healthcare services, and self-driving cars in smart cities [24] have stringent high QoS and throughput requirements. In order to support such applications in real-time by processing data and end-user requests locally, cloud services need to be decentralized and must be brought to very close proximity to the hardware via edge computing nodes. Motivated by these methods, we propose the new architecture that keeps a partial Forwarding Information Base (FIB) table in the local router for network performance while guaranteeing coherent routing with the support of the cloud.

Innovations such as Cisco Cloud Application Centric Infrastructure (Cisco Cloud ACI) [25] help to blur the distinction between the private data-center SDN and public clouds. Such architecture offers greater transparency between private SDN networks and public clouds to leverage internal SDN to access data and communicate with on-premises applications through secure interconnect for multi-cloud environment. Moreover, SDN platforms for cloud services like Meridian by IBM [26] and PDSDN [27] adopt the service-level network model with connectivity and policy abstraction schemes to process cloud tenants' requirements and priorities. SDN controllers also come handy in this regard, as they can update the forwarding tables in the switches (via southbound APIs) while the cloud manager (e.g., OpenStack [28]) is provisioning new computing resources by running new virtual machines (VMs) on a hypervisor. In this context, to support heterogeneous development of the control plane functions, Control Orchestration Protocol (COP) [29] was proposed to abstract the implementation of the proprietary functionalities. This protocol allows provisioning of end-to-end transport services among VMs across multiple network domains by enabling combined orchestration of cloud resources and network components.

In a vein similar to CAR's approach, a shift from the traditional monolithic SDN controller to a container-based microservices-oriented SDN controller [30] is gaining its momentum for more reliability against failures. Such microservices-based approach can be orchestrated using a conventional container orchestrator (e.g. Kubernetes [31]) coupled with a distributed event processor (e.g., Kafka [32] or gRPC [33]) and offers better maintainability and portability (potentially a disaggregated code repository). This overcomes the intrinsic weaknesses of a monolithic controller by requiring fewer components to deploy a functional controller core, isolating failed component quickly, and restarting only the affected micro-service without recompiling the entire controller [30]. Despite having microservice related safety concerns [34], potential migration of microservices to remote platforms is promising.

To design a new hybrid routing architecture such as CAR, these initiatives have encouraged us to borrow the core technology of SDN for bringing autonomic network function deployment in clouds by leasing the cloud's on-demand computing and storage resources.

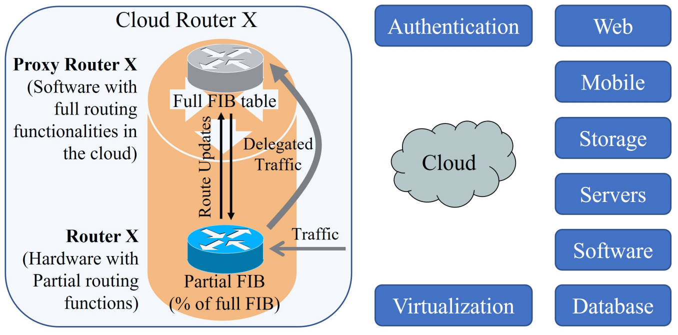


Fig. 2: CAR as a Cloud service

## III. CAR Architecture

We propose CAR as a new service offering from the cloud providers. Similar to the existing services such as storage services, database services, web services and many more (Figure 2), cloud providers will develop CAR as a new value-added service which ISPs can purchase per their needs. In this section, we detail the CAR architecture and the potential gains from it.

### A. Architectural View

An architectural view of a hybrid "CAR router" which combines a legacy hardware router with partial functionality and a software router with full functionality is illustrated in Figure 2. Similar to how virtual memory systems use secondary storage to keep full memory content, CAR uses cloud to implement the full functionality of local *Router X (RX)*, and keeps *RX* as 'active' while *Proxy Router X (PRX)* as 'passive'. The software *PRX* holds the full forwarding tables and performs the full routing computations, and is the default point of service for data and control plane functions that the hardware *RX* cannot handle. We anticipate that some of the control plane operations such as on-demand route computations due to failures or collection of flow-level simple statistics will still be done at *RX* similar to how recent OpenFlow switches account for the flow statistics and send them to controller. However, CAR will host heavy routing optimizations at *PRX*. With the recent growing interest in *Stateful Data Planes* [35] which introduces new architectures like OpenState [36] and FAST [37], we propose to partially delegate the data plane activities to the controller, sitting in the cloud.

As indicated in Figures 1 and 2, CAR aims to find a middle-ground between a *pure local approach* that targets to *scale* router performance and a *completely cloud-based approach* for seamless and highly *flexible* routing services. Rather than establishing a clean separation of control and data planes in legacy SDN, CAR argues for a hybrid opportunistic approach (Figure 2). Particularly, CAR places most of the heavy control plane functions (e.g., peering establishment) in the cloud while the most important and urgent control functions (e.g., response to failures) stay in local router. Intertwined with this control plane placement, most of CAR's data plane stays at the local router while a small portion of the data packet forwarding gets delegated to the cloud. CAR follows a homogeneous approach like RouteFlow [38]. The key differences are, in CAR, a) the controller sits on the cloud, and both *RX* and *PRX* can act as separate entities, and b) *PRX* is capable of establishing BGP peering with others by itself. While Figure 2 outlines the bare-bone architecture of CAR, the implementation details of various CAR protocols are to be explored. CAR service providers may design their own creative solutions utilizing single VM vs. multiple VMs, single cloud vs. multiple clouds to implement sub-functionalities as they see fit. We briefly discuss some design opportunities later in Section IV-B.

*1) Control Plane:* To scale computational complexity of routing, CAR advocates for delegating control plane tasks to cloud to the extent possible and perform large-scale routing computations by exploiting the cloud's cheap parallelism speedups as well as centralized role in traffic engineering. Though earlier proposals aimed to exploit parallelism in routing by modularizing a router into many parallel working nodes [39], cloud computing offers extensible resources beyond what can be offered locally. Yet, many routing problems require on-demand fast computations (e.g., calculating backup paths) which naturally fits to the CAR's approach of keeping a small portion of the control plane at *RX*.

*2) Data Plane:* Commercial routers provide highly optimized data plane implementing forwarding operations in extremely fast ASIC circuits and forwarding tables in custom TCAM memories. Many existing proposals suggest more programmable packet classification and forwarding function definitions at data plane [40]. As for CAR, packet classification, flow descriptions and corresponding forwarding actions (e.g., traffic shaping, differentiated service mechanisms, packet filtering, provisioning) are defined as "movable states" computed by control plane. CAR's data plane consists of virtualized low layers, movable upper layer states, and forwarding components used to compose both hardware and software level customizable data plane functions. In comparison to legacy SDN approach to routing, CAR's key difference is to move a small portion of the data plane forwarding actions to the cloud rather than keeping all of them at *RX*. This means that some of the data packets will be forwarded to the *PRX* at the cloud for further forwarding towards their destinations. This is a major departure from the legacy SDN where data packets and the task of forwarding them are strictly kept at local switch.

## *B. Principles*

Following the Amdahl's Law, CAR treats the router hardware as a precious resource that should only focus on the most frequent or important routing functions and offload the rest to the cloud. The following two principles should be followed when applying CAR to a routing problem:

**CPU Principle:** *Keep Control Plane Closer to the Cloud. RX* should be designed in such a way that it can offload computation intensive but on the same time not-so-urgent control plane tasks (such as BGP table exchange, full-fledged shortest-path calculations for traffic engineering optimization) to cloud to the extent possible. Our early prototype [41] attained 5 times reduction in CPU utilization burst size while establishing a BGP peering by exchanging a very small portion of table at RX and the full exchange at *PRX*s of two CAR-enabled BGP routers.

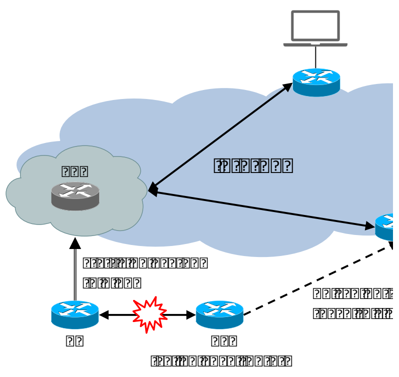
**Memory Principle:** *Keep Data Plane Closer to the Router. RX* should be designed to handle most of the packet forwarding operations locally. CAR designer can follow this memory principle simply by periodically identifying the most frequently used prefixes and storing a copy of them in router memory to deal with most of the forwarding lookups. If this can be optimally designed, *RX* will have to delegate few lookup requests to *PRX* which should keep the entire set of prefixes and will act as the default point of failure.

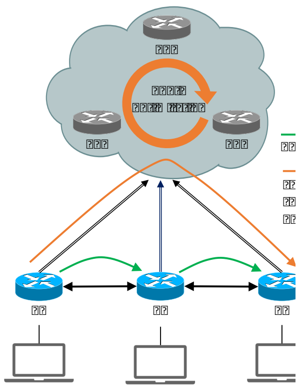
## *C. Function Placement*

The fundamental technical challenge a CAR designer faces is how to place the routing data and control plane functionality between *RX* and *PRX*. In general, placing some of the router functions at a remote location like the cloud will degrade the router's performance, i.e., the overall latency a data packet experiences will increase. Yet, with strategic placement, it is possible to keep the overall performance while handling much larger loads. For instance, by storing only 25% of FIB, *RX* can serve 99% of the traffic [42] without having to use *PRX* at the cloud, which means better performance. If the lookup at partial FIB table at *RX* is a miss, the packet will either be cached or delegated to *PRX*. If the *cloud-router delay*, , is small (e.g., 50ms) or the local buffer is too small to temporarily keep the packets of this flow, it is best to delegate the packet to *PRX*. But, if it is imminent that many more packets will arrive on this flow and continuos delegation of it to *PRX* will make too many customer packets to experience increased delay, it may be best to resolve the miss in a manner similar to the page fault handling (OpenFlow controllers follow this approach) and still process the data traffic at *RX*. Implementation of such router-cloud integration will require a more detailed analysis involving all the key parameters, such as , *RX* buffer size, flow rate, cloud response time, and priority of the flow.

*To-delegate or not-to-delegate.* In general, if is too high, CAR designer should not delegate to the cloud and keep more routing functions at the router to achieve higher overall efficiency. For a limited hardware router, other factors such as rates of traffic flows and router's buffer size will play role in identifying which flows to delegate. The flows delegated to cloud will receive service with larger delays and higher loss; and thus, keeping a fair treatment of data flows is a challenge CAR faces.

*Adaptive tuning to exploit locality patterns in traffic.* The key indicator for fruitfulness of CAR is whether it is possible to achieve a similar performance with smaller router hardware resources. This requires adaptive tuning of caching and delegation of router's functions for different traffic patterns and situations. Just like the virtual memory does not pay off if there is no locality, the benefit of CAR will highly dependent on the effectiveness of this adaptive tuning.


(a) Traffic reroute using cloud-based


(b) Shortcut through single cl dor

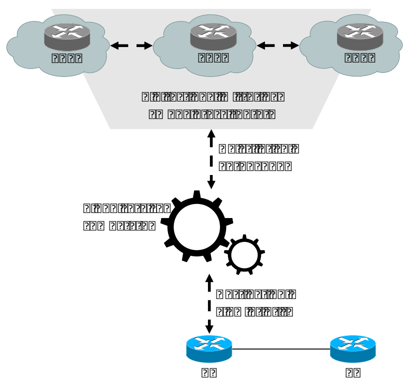

(c) Hybrid Design using Multiple Cloud Vendors

Fig. 3: FIB table lookup scenario using CAR

## IV. Hybrid Cloud Integration Potentials

### A. Establishing Robustness via Multi-Cloud Designs

Migrating routing functions to the cloud has sizable risks due to uncertainty of the cloud response times and hazy reliability to the cloud services in general. Yet, it may also help in improving the reliability of the router hardware as well. We discuss a few such scenarios.

*1) Failure-Triggered Traffic Delegation to PRX:* In case of a major link failure, significant amount of traffic should be rerouted without deteriorating the service quality levels for the remainder of the network. There has been extensive research on analyzing link failures, redundancy/over-provisioning models and architectures to achieve greater network resilience [43]. Similarly, upon a failure on a link to a neighbor router, *RX* can forward/delegate the affected traffic (which would normally go to the neighbor) to *PRX* as shown in Figure 3a. Such delegation could potentially be done in a manner seamless to other routers. However, potential inconsistencies will have to be considered and prevented. For example, *PRX* should not be somehow forwarding the traffic back to *RX*. One simple way of assuring this does not happen is to maintain a separate lookup table at the cloud node where *PRX* is located and check if *PRX* is the next hop for the destination prefix in question – in a manner similar to BGP's poisoned reverse. Yet, this only solves one-hop inconsistencies and consideration of policy issues will make this task more complicated. However, it will be possible to manage these complications by carefully organizing which prefixes are to be allowed for such failure-triggered traffic delegation. Also, CAR service providers are responsible for ensuring the availability of *PRX* by mitigating any cloud computing infrastructure failure occurring in *a*) application layer, *b*) servers hosting the CAR application, or *c*) inter-network connecting the physical servers. Failure prediction models [44] along with existing resiliency techniques [45] can be implemented in order to prevent any service interruption at *PRX*. Nevertheless, similar to legacy router, CAR router must have a default gateway by configuration. Due to hybrid nature of CAR, both *RX* and *PRX* need to store this gateway information separately. Despite taking all the precautionary measures, when the cloud where *PRX* is residing fails and at the same time, *RX* needs to consult with *PRX* about a packet's next hop, this would mean that the entire CAR router is failed and the default gateway at *RX* will have to be invoked for those packets being sent to this failed CAR router. Figure 3a does not describe this particular scenario since this can be handled by configuring BGP default route in *RX*. In general, delegating data plane forwarding tasks to a remote platform is a new feature being introduced by CAR and needs reworking of some of the existing configuration and policy practices.

*2) Migration of CAR Routers via Movable States:* Another interesting aspect of CAR is to decouple lower layer of network configuration with the states of upper layers by migrating some of the control and data plane functions to a third-party cloud service. This separation enables definitions of movable soft states which can define forwarding information for flow descriptors, service provisioning for differentiated services, security settings or even enhanced packet forwarding functions. One perspective of this "CAR Router state" is the mapping between these lower layer configurations and upper layer state. Currently, virtualization and tunneling technologies offer wide-range of flexibilities, as also required for achieving our CAR architecture. Telecommunication vendors also have various consortiums for developing common standards and interfaces to enable such mechanisms, e.g., [46]. Also, significant research work has been done on virtualized service mirroring and migration [47] especially for virtual machines. However, these technologies are still bound to the limitations of both hardware and software protocols.

Once feasible, such decoupling can be used to achieve "movable states" of provisioned services or forwarding plane among virtualized hardware. This can become especially important for Virtual Network Operators (VNOs), switching infrastructure providers and infrastructure owners to quickly recover after node failures. CAR architecture can provide quick access to the cloud storage and computation capabilities to compute, store and retrieve these movable states. Such virtualization of router states will be tremendously helpful in configuring, maintaining and optimizing large networks.

### B. Intra- and Inter-Domain Routing Optimizations

When CAR attains sufficient penetration and gets deployed by many autonomous systems (ASes) and cloud providers, several routing optimization opportunities will emerge. Cloud providers will, then, be able to perform centralized optimizations as they will be offering cloud assistance services to multiple routers in the same AS and routers belonging to different ASes. We outlined a few of such cases where CAR will be very beneficial in optimizing existing routing.

*1) Shortcuts Through Cloud:* One obvious benefit of delegating some of the packets to *PRX* is that the packet could be easily forwarded to the next hop on its path if the next hop router is being served by the same cloud provider. Figure 3b portrays such a hypothetical situation which can easily arise among *PRX*s belonging to the same AS, since it is very likely that routers of an AS will be receiving cloud assistance service from the same provider. Inter-AS shortcuts can be taken if the cloud vendor is providing service for routers from different ASes. Studies showed benefits of such cloud-based data transfers for higher level performance as well [48]. These optimizations become possible with CAR because data plane functionality is delegated to clouds. Legacy SDN designs strictly keep data plane to local router and only place control plane to a remote platform, and this clean separation of control and data planes disallows multi-hop optimizations within the data plane.

*2) Traffic Engineering:* Having control plane of many routers located in the same cloud, it becomes possible to implement traffic engineering techniques much more advanced than today's. Existing traffic engineering involves long timescale (e.g., days or weeks) re-configurations of routers and intuitive automated responses to failures. Complex load balancing tasks are done by human admins and require heavy computations of how to set layer 2 (e.g., MPLS) paths. With CAR, the cloud provider will be able to observe many routers at the same time and perform short timescale (e.g., tens of minutes) engineering of the traffic by using abundant computation and storage of clouds. It will be reasonable to run machine learning algorithms to detect emerging patterns in the traffic at short timescales and re-configure *PRX*s (and in turn *RX*s) accordingly. Similarly, in inter-domain traffic engineering, some of the pitfalls (e.g., misconfiguration of MED) could be prevented if ASes agree to an "inter-domain manager" offered by the cloud provider.

*3) Cloud as An Exchange Point or Mediator:* The performance of packet forwarding depends heavily on inter-AS negotiations. Mahajan *et. al* [49] explored a negotiation framework *Nexit* based on stable and efficient routing among neighboring ISPs. It is observed, if ASes share little information with each other, optimizing the routing becomes much easier as both of them would have a more comprehensive knowledge of the network. Kotronis *et. al* [50] proposed a routing model where control plane is logically centralized and managed by a trusted third party. According to them, the more ASes will delegate their control plane tasks to this third party, who has expertise in routing functionality and has a well-trained infrastructure to offer better management, the easier it will be to provide efficient reconciliation among these ASes. As of now, we can confidently claim, cloud can be treated as the best third party to offer such a centralized scheme.

Exploiting birds eye view of multiple ASes topology, cloud provider can easily detect and resolve policy conflicts, monitor and troubleshoot inter-domain routing problems and even come up with ingenious design solutions. This habit of delegating partial control logic will essentially create clusters in cloud. At this point, we want to ensure that AS-cluster [1] will be different from AS-confederation (defined in RFC 5065 [51]). While delegating control plane functionalities to cloud, it is assured that, cloud provider will correctly map individual AS's Service Level Agreements (SLA) into policy, enforce scalability issues and monitor misconfiguration. Cloud provider will also be responsible for keeping the confidentiality of AS specific topology, network loss, delay and bandwidth utilization.

### C. Flow Management

In a multi-tenant environment like the cloud, where users implement their own security measures (e.g., Intrusion Detection & Prevention Systems, Deep Packet Inspection, Virtual Private Networks, and Moving Target Defense) in their private logical infrastructure, leveraging SDN controller's holistic view seems natural to enable firewall functionality. Due to the complexity involved in generating distinctions between the (tenant) subnetworks, despite their (possible) overlapping address ranges, installing flow rules in the control plane leaves open for potential conflict and may impede the effectiveness of security infrastructure [52]. Following existing security policy analysis frameworks for cloud or SDN-based firewalls that work in collaboration with controller to provide a centralized security framework [53], CAR providers can implement frameworks that can translate the high-level security policies defined in the cloud into low-level flow rules deployed directly in the local routers' data planes. Since *PRX* relies greatly on restricted network topology and entry points, implementing traditional firewall functionalities at *PRX* will offer effective security protection from malicious flows and DDoS attacks.

A dynamic flow management framework, capable of changing flow control rules on the fly, is also needed to maintain a disruption-free transport layer functionality; as recent studies [54] show that traffic loss can occur on SDN hardware when active traffic flow rule is modified during an ongoing flow transmission. Apart from restricting the data plane to simply forward packets and keep statistics, we anticipate CAR providers will gradually take the advantage of the hybrid nature of CAR for scheduling application-specific network flows [55] by executing code at precise locations to optimize bandwidth and achieve reduced latency.

## V. Data Plane Delegation Concerns

### A. Picking the Best Cloud

Studies [56], [57] showed that it is possible to pick the best cloud provider for one's location for different application-desired metrics such as response time or service price. This

[1]AS-cluster will compose those ASes who are willing to trust their internal policy with cloud provider while preserving the right to manage their own privacy and integrity, business identity and policy-shaping capability.

presents a great opportunity to help our CAR archite
via consideration of multiple clouds to establish reliabili
*PRX*. As shown in Figure 3c, we can mirror *PRX* at mu
cloud providers and dynamically select the "best" one
time and based on the task at hand. An intermediary c
be designed to cope with the variations in the cloud se
quality. For example, when a packet is to be delegated to
such intermediary can balance various parameters like (i
importance of the traffic flow the packet belongs to, (ii
closeness of the candidate clouds to the destination o
packet, and (iii) the service prices of the candidate clouc

The intermediary could be implemented at *RX* or a
cated machine or a server in proximity of *RX*. It may eve
possible to migrate this intermediary to a cloud provider tl
willing to provide certain performance assurances in resp
time, which of course would entail costlier pricing.
type of multi-cloud framework requires efficient mirr
implementation of *PRX*s among multiple clouds, which c
be done with legacy mirroring protocols. This multi-cloud approach provides opportunities beyond failure management and allows various optimizations among the backend cloud providers depending on the traffic characteristics, e.g., its destination or priority. Existing SDN implementations are looking for solutions to increase the reliability of the SDN controller and the link(s) between the hardware routers and the controller [58]. If significant portion of the control plane can be migrated to multiple third-party cloud providers over a public Internet connection, vulnerability of the controller and the router-controller link to failures and attacks can be mitigated.

### B. Attaining Loop-Freeness

Due to the development of SDN designs with clean control plane separation, a prominent issue arises is whether there can be such a situation when a packet never reaches its final destination because of the loops [59]. It is always adverse to have a network inconsistency that has been generated due to the complex chaining of instruction. Similar concerns apply for the proposed hybrid SDN architecture. While designing such a scheme, network designers should be careful about possible inconsistencies, particularly when the data traffic delegated to *PRX* is using public Internet routing. Considering the existence of multiple hybrid SDN routers on a path, Figure 4 explores two possible loop scenarios that may arise due to data traffic delegation:

*1) Scenario A: While Reaching PRX From RX:* While delegating data traffic towards *PRX*, a loop may emerge if the Internet routing chooses *RX* as the next-hop. This scenario is not possible unless there is an inconsistency in the Internet routing. Assuming that the delegated traffic will most likely to travel through an IP tunnel, the shortest path towards the destination of that tunnel (i.e., *PRX*) will not traverse *RX* in steady-state routing.

*2) Scenario B: Between PRX and Destination:* Once delegated to *PRX*, it is *PRX*'s responsibility to forward the data traffic towards its destination, , via the appropriate shortest-path Internet routing. Such *delegated* traffic may or may not

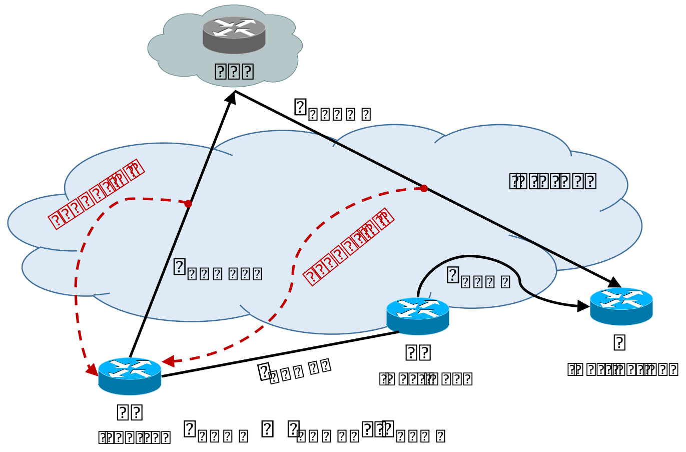


Fig. 4: Possible loop scenarios

visit the next hop router, *RY*. Regardless, such traffic may encounter a loop if the distance from *RX* to the destination, , is shorter than the distance from *PRX* to the destination, . Further, a loop exists if the path from *PRX* to the destination, , includes the path from *RX* to the destination, . So the following conditions have to be satisfied in order for a loop to form on the path from *PRX* to the destination:

1) *Necessary Conditions:*
   a)
   b)
2) *Sufficient Condition:* includes

Observe that if the delegated traffic visits another CAR router on the path from *PRX* to the destination, the loop forming conditions above are recursively applicable.

One approach to prevent the loops from *PRX* to the destination is to *only* allow delegation of prefixes to *PRX* for which *PRX*'s forwarding table has strictly less cost towards the destination, i.e., . This inequality condition will ensure that the necessary condition 1a for the loop is never met. Yet, such a preventive design may be too restrictive in terms of caching of prefixes in *RX* and limit traffic engineering possibilities. The necessary condition 1b is tighter and allows a more relaxed delegation scheme. Both of these necessary conditions can be reinforced at the *PRX* as it has access to all the necessary values, i.e., , , and . This loop prevention approach can be implemented in an SDN setup by adding a binary flag for each flow entry at the *RX*, indicating whether or not the data packets matching to that flow are delegatable. This flag can be set by the *PRX* when it is installing/updating the flow entry to the flow table(s) at *RX*.

In a more flexible design, the necessary conditions can be allowed but the sufficient condition can be avoided. However, avoiding the sufficient condition requires checking whether or not the path from *PRX* to *D* includes the path from *RX* to *D*. Since the full path information may not be available or verifiable, it is not possible to guarantee that this approach will not result in a loop, though practical methods can be devised. In the case of BGP, the sufficient condition can be checked by comparing AS-PATH attributes of the PRX-to-Destination and RX-to-Destination paths. Since the AS-PATH attributes are not verifiable, it still does not guarantee loop avoidance but provides a strong method to check for the existence of the condition.

In the above scenarios, multiple hops are allowed between *RX* and *PRX*. If the delegation from *RX* to *PRX* is a single-hop design (i.e., there is no other autonomous system or router in between *RX* and *PRX*), we can easily avoid loops by keeping a separate lookup table in the cloud where *PRX* resides (similar to BGP's poisoned reverse) and make sure the next-hop is not *RX*. Further, having multi-cloud designs (as in Figure 3c) allows implementation of staged policies where delegations that are safer can be chosen on the fly, e.g., if is satisfying the conditions 1a and 1b but is not, can be preferred.

## VI. Reducing Forwarding Table Size Using CAR

A well-known issue with core BGP routers is their forwarding table (FIB) and routing table (RIB) sizes. Several studies observed temporal (bursts of packets in the same flows) and spatial (few popular destinations) locality in data packet traffic [42]. This means, even though most of the destinations will be looked up very infrequently, they will keep occupying the routing table. A key motivation for CAR is to leverage these locality patterns and delegate the less used majority to the cloud while keeping the more used minority at the router.

The idea is to store only partial FIB at the *RX* and delegate packets to the *PRX* if a miss occurs during the lookup at the partial FIB. The *PRX* will store the full FIB, and thus will be able to handle any misses at the *RX*. As shown in Figure 5a, one can implement this relationship between *RX* and *PRX* via tunnels or dedicated TCP sessions. CAR handles FIB lookups in a hierarchical manner, as shown in Figure 5a. For instance, some packets will be handled completely via hardware lookups (e.g., packets destined to 8.8/16 in the figure), some via software lookups at the router (e.g., destined to 72.24.10/24), and some via lookups at the cloud (e.g., destined to 72.36.10/24). Similar to traditional cache organizations, the lookup will be delegated to one level up in the hierarchy if a miss occurs. In general, the placement of prefix entries to the different levels of this CAR framework is not an easy task [60], [42]. It involves several dynamic parameters such as lookup frequency of prefixes and importance of prefixes due to their contractual value. The positive factor is that high locality patterns exist in these parameters. Delegated prefixes will suffer from additional delays, and a key issue will be to establish an acceptable fairness across prefixes.

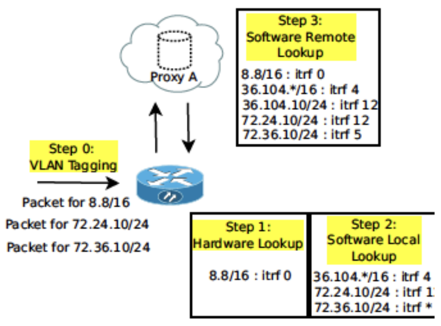
To make initial observations, we developed an Emulab [12] prototype of the concept shown in Figure 5a using a two-level hierarchy, one was the hardware lookup at the *RX* and the other was the software remote lookup at the proxy router *PRX*.

Figure 5b describes our experimental setup for the initial CAR router prototype with three next-hops. Although it is possible to implement existing FIB caching techniques in CAR, we prototyped a *crystal ball* approach where the local router has a holistic view of the entire FIB table and is assumed to know the future packets. At beginning of every second, before any incoming packets appeared, *RX*'s FIB cache was programmatically updated with the most frequently used prefixes for that particular second so that the router was always aware about the arriving packets and could redirect the packets to their appropriate destinations without any delay. We call this as *crystal ball* approach because of the router's knowledge about the future packets even before they arrive. This design gives us an understanding of the best possible gain from CAR in terms of FIB size reduction. To ease the measurements, we used one destination router. We used anonymous traffic traces collected by CAIDA [61]. For routers, we used Quagga, with each router having four available gigabit net interfaces. Links were configured as duplex with standard DropTail queues of size 100 Mb and a loss probability of 0.01. To feed the routing entries, we identified the subnets of all packets and assumed them to be of /24. We are using /24 subnet prefixes based on the observation [42] from RouteView traces. /24 prefix count increased from 60*K* to 140*K*, while /16 increased 7*K* to 10*K* and /8 increased from 18*K* to 19*K* which shows a much quicker increase compared to other prefix lengths. Then we took top % of the /24 prefixes with the highest frequency and varied from 1, 2, 5, 10, 15, 20. We feed only these most frequent subnets to the FIB table located in *RX*, while *PRX* had the entire list. Since we followed a *crystal ball* approach, we continued feeding the most frequently used subnets to *RX* at the beginning of every second before allowing any packets for that particular second to reach *RX*. Note that the cache hiding does not exist in this case as all prefixes are /24.

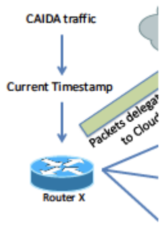
Figure 5c shows the percentage of packets being delegated to the cloud proxy as the FIB cache size at the local router increases. It is clear that, the trend is going downwards and if we use 20% of entire FIB table as cache, it will miss only 7.7% of all the packets. Yet, the gain is not as high as expected since earlier measurement studies showed that 10% of FIB accounts for more than 97% of the traffic and 1% accounts for more than 90% of the traffic [42]. In comparison, our crystal ball approach could only process about 87% of the traffic when FIB cache size was 10% and 60% of the traffic when the cache was 1%. This is probably because of the delay involved in updating the local router's FIB table. However, the significant insight from this experiment is the increase in the gain, because, the impact of delay reduces significantly if FIB cache size is kept above 15%.

Table I shows the number of packets forwarded via individual next-hops and time it took for those packets to reach the destination router if the FIB cache size is only 1% of the entire FIB. It also shows the number of packets delegated towards the cloud. Couple of observations from this table are % of dropped packets towards the next hops are quite high, 4.72 to 4.96. The trend was same for the rest of the experiment as well. This is because of the smaller size of buffer as well as the DropTail technique used in the Emulab machines. Figure 6a shows the CDF of time required for each packet to reach the destination, which shows a minimal unfairness. Most of the packets arrive in similar timings.

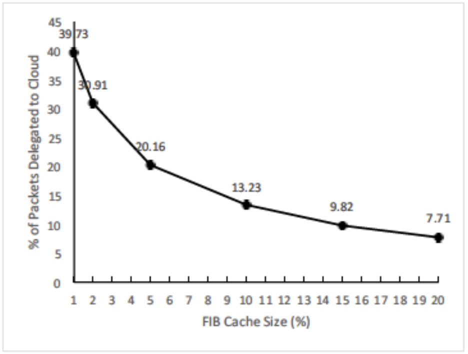
Figure 6b plots the CDF of packet delay for FIB cache size 10%. We observe an increase in the end-to-end delay when FIB cache size is increased, which is counter-intuitive. The



(a) Forwarding table lookup



(b

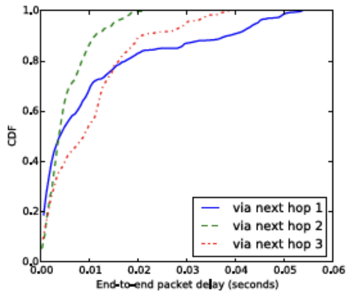


(c) Packets delegated to cloud

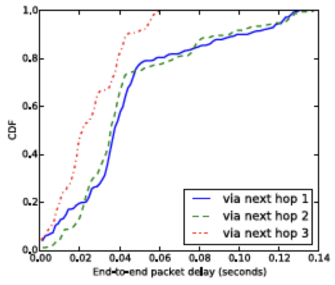
Fig. 5: FIB table lookup scenario using CAR

| NextHop | Packets Sent | Total Packets (%) | Packets Dropped (%) | Avg. Delay (ms) |
|---|---|---|---|---|
| NextHop-1 | 7,949,903 | 20.79 | 4.72 | 17.7 |
| NextHop-2 | 7,670,496 | 20.06 | 4.74 | 13.8 |
| NextHop-3 | 7,426,526 | 19.42 | 4.96 | 13.7 |
| Cloud | 15,191,706 | 39.73 | - | - |

TABLE I: FIB Cache Size 1% (Emulab Setup)



(a) FIB cache size 1%



(b) FIB cache size 10%

Fig. 6: End-to-end delay in CAR

reason for this increase in the end-to-end delay is the fact that the traffic experiences more delay from *RX* to the destination when there is more traffic taking that path. Delegation to *PRX* reduces the load on the path from *RX* to the destination router, and hence reduces the queuing delay on that path. This is an interesting finding and needs further exploration. In particular, in terms of the average end-to-end delay, the delay from *RX* to *PRX*, $d_{RX \rightarrow PRX}$, may be absorbed by the reduction in the delay from *RX* to the destination. Depending on the actual value of $d_{RX \rightarrow PRX}$ and the queuing delay on the path via *PRX*, the optimum FIB cache size can be tuned. This delay tradeoff between the actual path (taken by the packets processed locally at *RX*) and the "delegated path" (taken by the packets delegated to *PRX*) is a new knob CAR offers to router design.

## VII. CAR Prototype

To show the proof-of-concept of CAR architecture, we stripped off some of the complexity from the original design and developed a simple prototype of CAR using SDN. The top-level architectural overview of our prototype is portrayed in Figure 7, where *Source, NH1, NH2 (next-hops)*, and *PRX* all represent individual ASes and have their external BGP routers. *RX* implements SDN architecture internally, has an internal BGP speaker, and, from the BGP perspective, appears as a single AS. We used FRRouting (FRR) [62], an open-source routing protocol suite, on virtual machines, each running Ubuntu to act as BGP routers.

We use Ryu [63] controller to serve our objective as it has a BGP speaker library that we utilize. We develop our custom Ryu controller application that creates a BGP instance and maintains an iBGP session with the internal BGP speaker (a separate BGP router). We shall refer to this application as the *CARbgp* application. The internal BGP speaker's only task is to maintain the eBGP sessions and receive the BGP announcements from peering ASes. We use this separate BGP router for managing the external communications to prevent any direct interaction between the controller and peer networks. This design choice also helps us to keep the controller isolated and secured by not exposing it to the outside. Our custom application installs flow rules in the RX-switch, an Open vSwitch (OVS) so that only BGP control messages coming towards the internal BGP speaker are allowed through its data plane. Once the BGP sessions are established, external peers start announcing their prefixes, which are eventually received

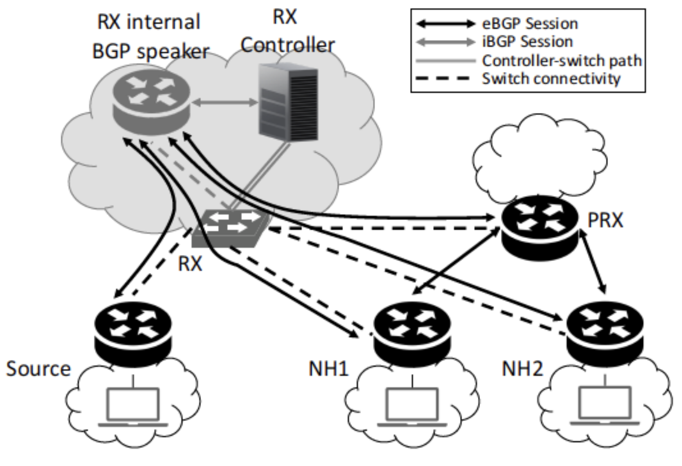


Fig. 7: CAR prototype using RYU SDN controller

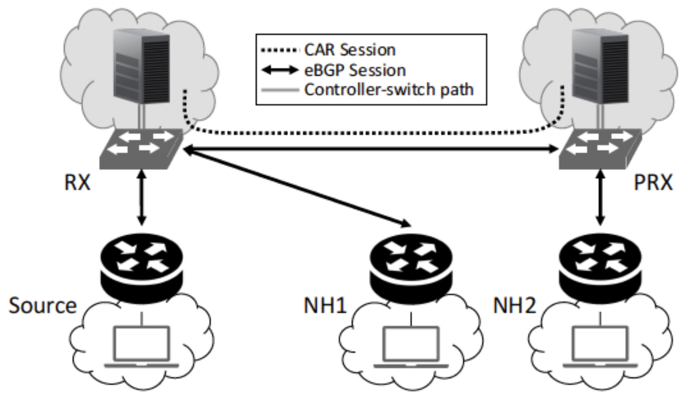


Fig. 8: CAR prototype using CAR protocol

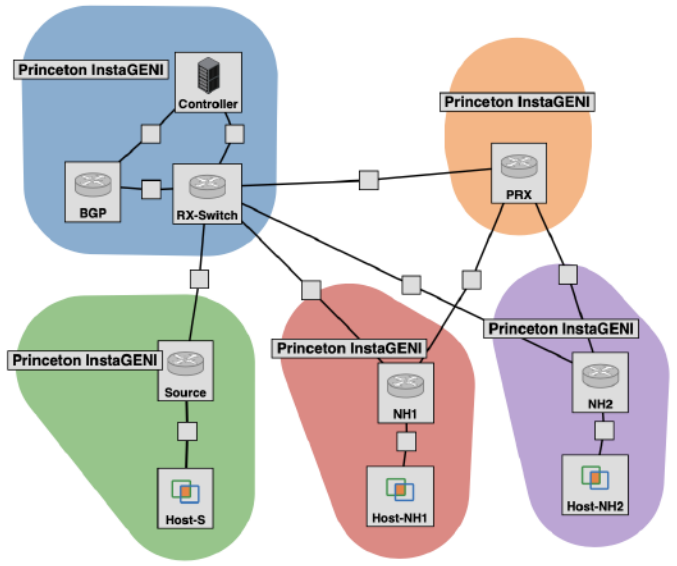


Fig. 9: CAR setup in GENI

by the controller and sent to *CARbgp* application to handle.

Each time *CARbgp* receives a new prefix announcement from an external peer, it adds a new flow rule in the switch for that. *CARbgp* maintains a one-to-one mapping of RX-switch ports and external peer routers connected to it. Thus, it is easy to add new flow rules to forward associated packets through its announcing router's connected port. To follow the routing activity, we also decrement the packet TTL by 1 and update the destination MAC address with the next-hop router.

Our main argument for CAR is that *RX* does not need to keep all the entries in its FIB. In a real-world scenario, if *RX* can not serve a prefix, it will forward that packet to *PRX*, as *PRX* will have the entire FIB table. The actual network architecture between *RX* and *PRX* will be similar to Figure 8. Since in CAR, *PRX* will have the exact copy of *RX*'s entire FIB, it is not mandatory that *NH1* and *NH2* have to be a direct BGP neighbor with both *RX* and *PRX*. However, in our stripped version of CAR, we made sure that all the BGP neighbors of *RX* also establish BGP peering relationships with *PRX*, so that *PRX* can at least have a direct path for the delegated packets to the destined AS.

For the experiment, we again employ the *crystal ball* behavior in *RX* similar to section VI. Again, we assume, the controller is aware of the most popular prefixes and will keep those prefixes in its FIB to support the most active flows and thus, ensuring smaller latency for overall traffic. We vary FIB caching thresholds from one to 20 percent and for each case, we compare the aggregate delays the traffic experiences. *CARbgp* always keeps the statistics of flow entries in the flow table. During experiments, at the beginning, as new prefixes are announced, *CARbgp* adds a new flow rule in the *RX*'s flow table for each new prefix until its FIB cache size capacity maxed out, which, in this case, was the flow rule count.

## A. Experimental Setup in GENI

We set up our experiment in GENI [64], a federated testbed that allows obtaining computational resources from different US locations, and installs various customized software on VMs if requested. We reserved multiple *slices*[2] from the same *aggregate*[3], Princeton InstaGENI rack, and set up our topology (see Figure 9) there. Each slice represented one individual AS. It is also possible to use different slices from different InstaGENI racks (e.g., Cornell, Illinois, and Stanford) and stitch together.

We assigned one single host to each of *Source*, *NH1*, and *NH2* ASes. Both *NH1* and *NH2* announced a list of 100 unique prefixes, separately, while exchanging the BGP announcements. To implement this, we assigned 100 virtual IP addresses from different subnets to both *Host-NH1* and *Host-NH2* instead of adding 100 VMs. For example, *Host-NH1* was connected to the *NH1* BGP router and represented the IP block of 192.168.1.0/24 to 192.168.99.0/24, while *Host-NH2* represented 192.168.100.0/24 to 192.168.199.0/24 networks.

To make our analysis more comprehensive and pragmatic, we used the *Anonymized Internet Traces* collected by CAIDA's *EQUINIX-NYC monitor* [66] during December 2018. We identified the unique IP addresses, and the packet counts destined towards each of them from these anonymized traffic traces. We sorted the IP addresses in descending order of their packet counts and clustered them into 199 bins. We then mapped each group one-to-one, sequentially, with one item from those as mentioned above, 199 distinct subnet IP addresses list. As a result, most popular IP addresses were grouped and are positioned at the beginning of our list of virtual IP addresses. For instance, there were 557,246 unique IP addresses in this particular traffic trace, and the most popular 2,801 IP addresses were grouped to be represented by 192.168.1.1. The rest were computed in a similar manner. We also accumulated the packet counts per group and treated that number as the incoming packet count for the associated virtual IP address. But, the numbers were too high for the first few IP addresses as they were the heavy-hitters. Thus, we scaled down all the numbers by dividing them with the smallest number among

[2]A *slice* is the context for a particular set of experiments and contains reserved resources and the set of GENI users who are entitled to act on those resources [65].

[3]An *aggregate* is a software server that provides resources to clients based on the GENI aggregate manager API. The *aggregate* may provide virtual 'sliced' or non-virtual 'whole' resources to customers [65].

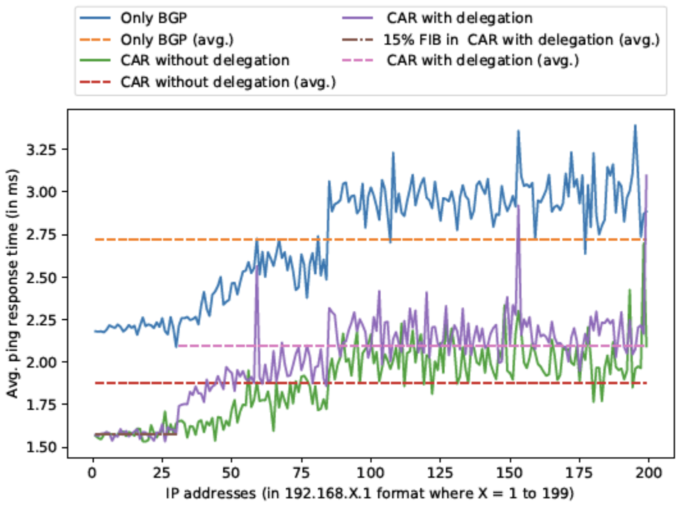
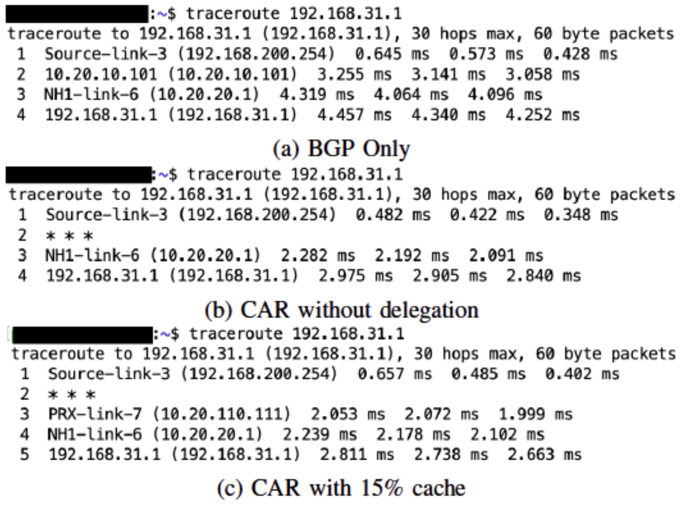


Fig. 10: Average ping responses of IP addresses



```
[redacted]:~$ traceroute 192.168.31.1
traceroute to 192.168.31.1 (192.168.31.1), 30 hops max, 60 byte packets
 1  Source-link-3 (192.168.200.254)  0.645 ms  0.573 ms  0.428 ms
 2  10.20.10.101 (10.20.10.101)  3.255 ms  3.141 ms  3.058 ms
 3  NH1-link-6 (10.20.20.1)  4.319 ms  4.064 ms  4.096 ms
 4  192.168.31.1 (192.168.31.1)  4.457 ms  4.340 ms  4.252 ms
```

(a) BGP Only

```
[redacted]:~$ traceroute 192.168.31.1
traceroute to 192.168.31.1 (192.168.31.1), 30 hops max, 60 byte packets
 1  Source-link-3 (192.168.200.254)  0.482 ms  0.422 ms  0.348 ms
 2  * * *
 3  NH1-link-6 (10.20.20.1)  2.282 ms  2.192 ms  2.091 ms
 4  192.168.31.1 (192.168.31.1)  2.975 ms  2.905 ms  2.840 ms
```

(b) CAR without delegation

```
[redacted]:~$ traceroute 192.168.31.1
traceroute to 192.168.31.1 (192.168.31.1), 30 hops max, 60 byte packets
 1  Source-link-3 (192.168.200.254)  0.657 ms  0.485 ms  0.402 ms
 2  * * *
 3  PRX-link-7 (10.20.110.111)  2.053 ms  2.072 ms  1.999 ms
 4  NH1-link-6 (10.20.20.1)  2.239 ms  2.178 ms  2.102 ms
 5  192.168.31.1 (192.168.31.1)  2.811 ms  2.738 ms  2.663 ms
```

(c) CAR with 15% cache

Fig. 11: Sample traceroute from Host-S (192.168.200.1) to 192.168.31.1

them. Finally, for measuring the end-to-end delay, we sent *ping* messages from *Host-S* to each virtual IP address using its corresponding incoming packet count.

Figure 10 plots the average ping response time for the IP addresses. We compare three design choices: *i)* using only BGP routers, *ii)* using CAR architecture, but keeping the full FIB in RX, and *iii)* using CAR architecture with 15% FIB in RX. Surprisingly, we noticed during the experiment that the average delay for each IP address was higher for the packets going through BGP routers only compared to what a packet experienced in CAR. "CAR without delegation" performed the best because the full FIB was available at the *RX* switch, and related flow entries were already put in the switch when the BGP announcements were received. When a packet arrived, the *RX*-switch easily forwarded that to the appropriate outgoing port without going through the additional delay of delegation to the *PRX*. "CAR with 15% delegation" showed an expected behavior. The cached prefixes (30, in this case) performed better than the delegated packets, which still outperformed the conventional BGP performance. We can see a hike in the average ping response time for the IP addresses on the right side of the figure. These IP addresses belonged to *Host-NH2* and were on a different virtual host. They always displayed a higher response time compared to *Host-NH1*. Since the physical resources are shared among the GENI users, we think, maybe, this specific machine was experiencing higher traffic volume from other researchers' experiments.

We kept the experimental setup in GENI the same for all three scenarios. For the "BGP only" experiment, the *RX*-switch acted as a simple layer-2 switch, and incoming packets from *Source* went to the internal BGP router residing inside *RX* and then traversed back via the *RX*-switch to either *NH1* or *NH2*. This incident can be one possible explanation for BGP's higher delay as a packet had to travel extra in this specific scenario. But, in the case of CAR architecture, the packets did not go back-and-forth between *RX*-switch and the internal BGP router except for the BGP control plane messages. During the BGP establishment process, *CARbgp* became aware of the prefixes and put related flow rules in the flow table of the switch for

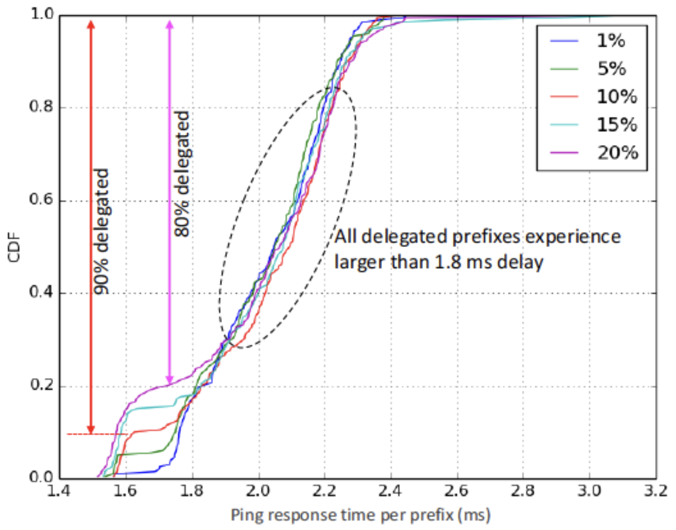


Fig. 12: FIB caching performance comparison: Average response time (round-trip delay) per destination prefix

them. So, the switch was able to make an immediate decision.

Figure 11a, Figure 11b, and Figure 11c show the `traceroute`s for the same destination (IP 192.168.31.1), from *Host-S* (IP 192.168.200.1), for the three different experimental setups. In Figure 11a ("BGP only" setup), *RX*-switch is not doing anything, and as such, from `traceroute`, there is no existence of it; rather, we can see the internal BGP router's IP address, which is residing inside *RX*'s network. Contrary to it, in the CAR version, we hide the IP address of our BGP router, as we can see from Figures 11b and 11c. Both the time, the packet travels through the same path, except for "CAR with 15% cache" delegates the packet to *PRX* instead of *NH1*. Since *NH1* and *PRX* are also BGP neighbors and are connected directly, *PRX* takes care of the packet and forwards it to *NH1*.

Moving one step further, we highlight the FIB caching effectiveness by comparing the total delay experienced for each destination prefix in Figure 12. For this purpose, we reran "CAR with *x*% cache" setup five times by varying the

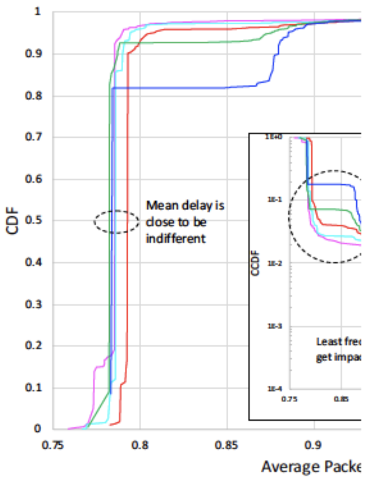


Fig. 13: FIB caching performanc experienced by packets

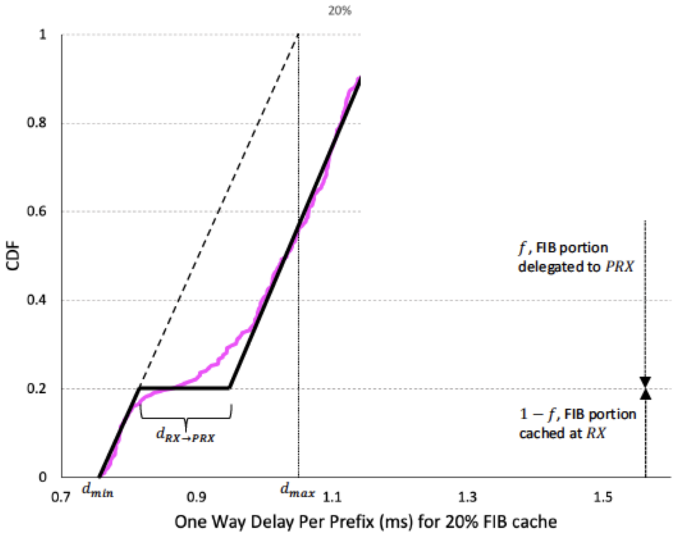


Fig. 14: Fitting a model for the per-prefix response time distribution

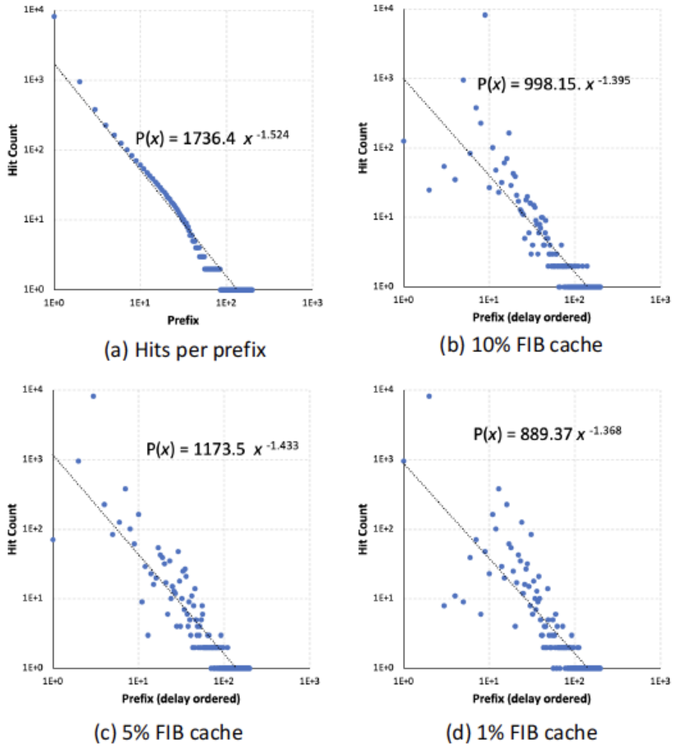


Fig. 15: Hit distribution across prefixes: (a) Hit distribution from the traffic trace. (b)-(d) Hit distribution per prefix while prefixes are ordered with respect to delay.

cache size from 1% to 20% ($x$ = 1, 5, 10, 15, 20). While above-mentioned Figure 11c only shows the `traceroute` for CAR setup with 15% FIB cache in *RX*, we plot ping response time per prefix for these five different cache-sized "CAR with $x$% cache" to show the overall performance in terms of delay in Figure 12. As expected, the prefixes that are cached at the *RX*-switch experience lower delay while all delegated ones experience larger delay. The transition from cached to delegated prefixes is clear with a threshold following the percentage of the prefixes being cached.

The difference in per-prefix delay does not directly translate to significant difference in the delay packets experience. Figure 13 clearly shows that FIB caching makes a minimal change in the mean delay the packets experience. When FIB cache size is increasing from 1% to 20%, the mean packet delay changes less than 0.5 ms. Because of the high locality of the traffic, the first few IP addresses which account for the majority of the traffic, and as long as the design can cache these few prefixes carefully, the overall performance will not degrade much. However reducing the FIB size does have a cost in terms of fairness. As shown in the Cumulative CDF plot, approximately 89 to 99 percentile as well as the tail above 99.9 percentile (i.e., less than 0.1%) of the packet delay distribution experience unfair treatment. Further, the least frequent packets (the ones in the tail) are the most impacted by the delegation to the cloud. This is a promising result since policies targeting these type of rare flows can be practically implemented, e.g., by detecting them and dedicating a space for them in the FIB cache.

## B. A Simplified Model of Delay in CAR Routers

Inspired by the results from our GENI prototype of CAR, we develop a simplified model of the delay experienced through a CAR router. Let $f$ be the fraction of FIB delegated to *PRX*. As can be seen from the per-prefix response time in Figure 12, the response time distribution roughly follows a Uniform distribution with a shift at the specific percentage corresponding to the amount delegated, $f$. This is illustrated in Figure 14 where $d_{min}$ and $d_{max}$ are, respectively, the minimum and maximum one-way delay (half of the response time) per prefix of the Uniform distribution. Let $t$ represent the per-prefix one-way delay and $F(t)$ represent the CDF of $t$. As observed in the figure, $F(t)$ has a tail, which the Uniform distribution model ignores. Since the tail is tiny (in terms of the percent of the prefixes) and the hit distribution across prefixes mostly follows the order of minimum to maximum one-way delay (which we will show shortly), ignoring the tail does not notably impact the accuracy of the average delay model.

In order to calculate the average delay, we need to factor in the hit distribution across prefixes. Let be the random variable representing the number of hits per prefix and be the PDF of . Due to well-known locality in traffic, follows power-law distribution, i.e., = where and are parameters. The traffic trace in our experiments also showed this behavior, as shown in Figure 15(a). To merge and for calculating the average packet delay, we need to know which prefix in the one-way delay distribution is receiving how many hits. In other words, we need the relationship between the order of prefixes in and . 's horizontal axis is ordered in delay. From our experiments, we observe that the hit distribution, , roughly follows the delay order of . Figures 15(b)-(d) show for various FIB cache size experiments when the prefixes are ordered in terms of delay, like . These plots show that it is safe to assume that the random variables and roughly follow the same order in terms of delay. Hence, we write the average packet delay as:

$$\bar{} = \int_{min}^{max} \tag{1}$$

Substituting power-law for and shifted Uniform distribution for with delegation, we obtain:

$$\bar{} = \int_{min}^{max} \frac{}{} \\ \int_{min\ \ max\ \ max}^{max} \qquad \frac{}{} \tag{2}$$

$$\bar{} = \int_{min}^{max} \frac{}{\Delta} \int_{max\ \ \Delta}^{max} \frac{}{\Delta} \tag{3}$$

where $\Delta =$ .

Since our hit distribution follows a power-law exponent very close to 1.5, we substitute = 1 5 and rewrite $\bar{}$ in closed form:

$$\bar{} = \frac{2}{\overline{} \quad \overline{}} \; \frac{2}{\Delta} \left( \frac{1}{\sqrt{\qquad \Delta}} \quad \frac{1}{\overline{}} \right) \tag{4}$$

By plugging in the empirical , , and values from our experiments, we calculate the average packet delay from the model in (4) and fit the model (i.e., minimize squared error) to our experimental average delay observations by tuning . Figure 16 shows the comparison of the model and our available experimental delay observations.

Beyond validation of the experimental observations, the closed form model in (4) provides insights into the limits of CAR routers' delays. The first term is the unavoidable delay as it is the part that comes from the delay in *RX* without any delegation to *PRX*. The second term is the delay component due to the delegation to *PRX*. When = 0, the second term cancels out, which verifies the intuition that the latency from *RX* to *PRX* should not be contributing to the delay when there is no delegation to *PRX*. More interestingly, taking the limit of $\bar{}$ as the variation in the one-way delay distribution across the prefixes, , goes to zero, we obtain:

$$\lim_{\Delta \quad 0} \bar{} = \frac{}{\overline{}} \left( 1 \quad \frac{}{} \right) \tag{5}$$

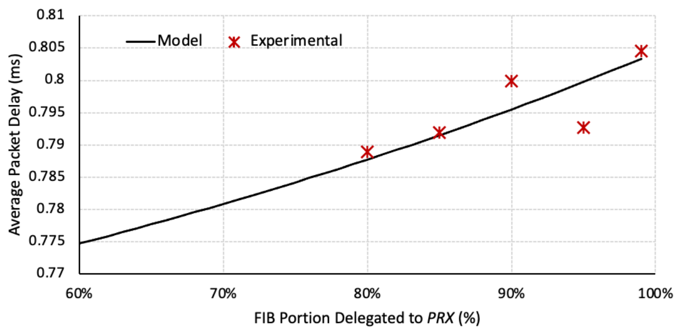


Fig. 16: Average packet delay model vs. experimental results

The regime in (5) happens when the minimum and the maximum delay to a prefix is the same, i.e., = . This extreme regime shows that the cost of delegation, i.e., the second term inside the parentheses in (5), depends on the ratio of the latency between *RX* and *PRX* to the minimum (or maximum) delay to a prefix. Hence, CAR designer can mitigate the cost of delegation by focusing on practices to reduce this ratio.

## VIII. Summary and Future Work

We introduced a new architecture, CAR, to address the growing complexity of routers and outlined its principles and key components, while navigating through possible benefits and implied challenges. We also presented our initial prototype on how CAR can help reduce FIB tables routers. Unlike the legacy SDN approach of clean slate separation of the control and data planes, CAR advocates on-demand seamless integration of the cloud's computational and storage capabilities into the core Internet routing. In a complementary approach, CAR hybridizes cloud computing and the SDN vision as integration between enterprise SDNs and clouds continue to deepen. Beyond bridging the gap between router hardware and software-based routing services, CAR allows opportunities to improve ISP backbones by integrated network monitoring and management such as *i)* resiliency to failures via cloud-based forwarding and reroute schemes, *ii)* efficiency via more centralized cloud-based optimizations of intra-domain traffic engineering, and *iii)* economic competitiveness via cloud-based on-demand service provisioning potentially going beyond domain borders.

Deploying CAR at inter-AS level will entail some changes to the existing policy routing practices. For a smooth transition, an AS could initiate a consulting phase by discussing about the best practice and informing CAR provider about its own requirements. Based on the observation of traffic flow, the CAR provider will first optimize the routing and deploy intelligent traffic handling in client's network. Later, in hand-over phase, AS will allow CAR provider to access its own FIB and finalize the process. Further models could be developed integrating CAR providers into the inter-AS routing architecture. In this paper, we addressed various architectural challenges of CAR, e.g., potential loops and reliability via multiple clouds. However, protocol implementations need to be explored with a focus on solving specific problems as part of the CAR architecture. Understanding the average delay of CAR routers under various conditions by expanding our simplified model will be insightful. In particular, stochastic delay models for multi-cloud CAR designs could show the potential and limitations of such advanced CAR architectures. Finally, it is worthy to explore the affect of network topology changes in CAR's failure handling performance and whether the protocols in the controller (e.g., utilizing P4 [67] to implement CAR protocol directly on programmable devices in addition to OpenFlow) can affect CAR's overall performance.

## REFERENCES

[1] P. K. Dey and M. Yuksel, "CAR: Cloud-assisted routing," in *Proceedings of IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Palo Alto, CA, November 2016, pp. 100–106.

[2] ——, "Hybrid cloud integration of routing control & data planes," in *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*, Irvine, CA, December 2016, pp. 25–30.

[3] M. Yuksel and H. T. Karaoglu, "Apparatus, system, and method for cloud-assisted routing," Patent USPTO US 9 634 922, 2017.

[4] X. Zhao, D. J. Pacella, and J. Schiller, "Routing scalability: an operator's view," *IEEE Journal on Selected Areas in communications*, vol. 28, no. 8, pp. 1262–1270, 2010.

[5] G. Huston, "BGP in 2019 – The BGP Table," January 2020. [Online]. Available: https://blog.apnic.net/2020/01/14/bgp-in-2019-the-bgp-table/

[6] P. C. da Rocha Fonseca and E. S. Mota, "A survey on fault management in software-defined networks," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2284–2321, 2017.

[7] O. TS-005, "OpenFlow Management and Configuration Protocol (OF-Config 1.1)," June 2012. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2013/02/of-config-1.1.pdf

[8] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[9] H. Wang, H. Xu, C. Qian, J. Ge, J. Liu, and H. Huang, "Prepass: Load balancing with data plane resource constraints using commodity sdn switches," *Computer Networks*, vol. 178, p. 107339, 2020.

[10] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic sdn controller assignment in data center networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2788–2801, 2017.

[11] P. K. Dey and M. Yuksel, "An economic analysis of cloud-assisted routing for wider area sdn," *IEEE Transactions on Network and Service Management*, 2019.

[12] "EmuLab – Network Emulation Testbed," "https://www.emulab.net".

[13] "GENI exploring future of the networks," http://www.geni.net/.

[14] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014.

[15] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, "Applying nox to the datacenter." in *HotNets*, 2009.

[16] Y. Yue, B. Cheng, X. Liu, M. Wang, B. Li, and J. Chen, "Resource optimization and delay guarantee virtual network function placement for mapping sfc requests in cloud networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1508–1523, 2021.

[17] H. Yu, J. Yang, and C. Fung, "Fine-grained cloud resource provisioning for virtual network function," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1363–1376, 2020.

[18] B. Ford, "Icebergs in the clouds: The other risks of cloud computing," in *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, Boston, MA, June 2012, pp. 453–466.

[19] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "SafeBricks: Shielding network functions in the cloud," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Renton, WA: USENIX Association, Apr. 2018, pp. 201–216. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/poddar

[20] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "Carpo: Correlation-aware power optimization in data center networks," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 1125–1133.

[21] K. Zheng, X. Wang, and J. Liu, "Disco: Distributed traffic flow consolidation for power efficient data center network," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2017, pp. 1–9.

[22] M. D. de Assunção, R. Carpa, L. Lefévre, and O. Glück, "On designing sdn services for energy-aware traffic engineering," in *International Conference on Testbeds and Research Infrastructures*. Springer, 2016, pp. 14–23.

[23] F. Spinelli and V. Mancuso, "Toward enabled industrial verticals in 5g: A survey on mec-based approaches to provisioning and flexibility," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 596–630, 2020.

[24] N. Hassan, K.-L. A. Yau, and C. Wu, "Edge computing in 5g: A review," *IEEE Access*, vol. 7, pp. 127 276–127 289, 2019.

[25] "Cisco Cloud Application Centric Infrastructure," 2019. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/solution-overview-c22-741802.pdf

[26] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an sdn platform for cloud network services," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, 2013.

[27] X. Du, Z. Lv, J. Wu, C. Wu, and S. Chen, "Pdsdn: A policy-driven sdn controller improving scheme for multi-tenant cloud datacenter environments," in *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 2016, pp. 387–394.

[28] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[29] A. Mayoral, R. Vilalta, R. Munoz, R. Casellas, R. Martínez, M. S. Moreolo, J. M. Fabrega, A. Aguado, S. Yan, D. Simeonidou *et al.*, "Control orchestration protocol: Unified transport api for distributed cloud and network orchestration," *Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A216–A222, 2017.

[30] D. Comer and A. Rastegarnia, "Toward disaggregating the sdn control plane," *IEEE Communications Magazine*, vol. 57, no. 10, pp. 70–75, 2019.

[31] "Kubernetes: Production-grade container orchestration," https://kubernetes.io/.

[32] "Apache kafka," https://kafka.apache.org/.

[33] "grpc: A high performance, open source universal rpc framework," https://grpc.io/.

[34] D. Yu, Y. Jin, Y. Zhang, and X. Zheng, "A survey on security issues in services communication of microservices-enabled fog applications," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 22, p. e4436, 2019.

[35] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful sdn data planes," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701–1725, 2017.

[36] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 44–51, 2014.

[37] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, "Flow-level state transition as a new switch primitive for sdn," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 61–66.

[38] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. Corrêa, S. C. De Lucena, and M. F. Magalhães, "Virtual routers as a service: the routeflow approach leveraging software-defined networks," in *Proceedings of the 6th International Conference on Future Internet Technologies*, 2011, pp. 34–37.

[39] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "Routebricks: exploiting parallelism to scale software routers," in *Proc. of the ACM Symposium on Operating Systems Principles (SOPS)*, 2009, pp. 15–28.

[40] M. B. Anwer, M. Motiwala, M. b. Tariq, and N. Feamster, "SwitchBlade: A platform for rapid deployment of network protocols on programmable hardware," *SIGCOMM CCR*, vol. 40, pp. 183–194, August 2010.

[41] H. T. Karaoglu and M. Yuksel, "Offloading routing complexity to the cloud(s)," in *Proceedings of IEEE ICC Workshop on Cloud Convergence*, Budapest, Hungary, June 2013.

[42] C. Kim, M. Caesar, A. Gerber, and J. Rexford, "Revisiting route caching: The world should be flat," in *Proceedings of the 10th International Conference on Passive and Active Network Measurement*, Seoul, Korea, 2009, pp. 3–12.

[43] M. Menth, M. Duelli, R. Martin, and J. Milbrandt, "Resilience analysis of packet-switched communication networks," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 1950–1963, December 2009.

[44] J. Gao, H. Wang, and H. Shen, "Task failure prediction in cloud data centers using deep learning," *IEEE transactions on services computing*, 2020.

[45] C. Colman-Meixner, C. Develder, M. Tornatore, and B. Mukherjee, "A survey on resiliency techniques in cloud computing infrastructures and applications," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2244–2281, 2016.

[46] "TM Forum IPsphere Framework," "http://www.tmforum.org/ipsphere".

[47] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford, "Live migration of an entire network (and its hosts)," in *Proceedings of HotNets*, 2012.

[48] A. Kuzmanovic, "Skynet: A cloud-hopping data transfer architecture," in *Proceedings of IEEE Annual Computer Communications Workshop (CCW)*, Cape Cod, MA, October 2011.

[49] R. Mahajan, D. Wetherall, and T. Anderson, "Negotiation-based routing between neighboring ISPs," in *Proceedings of USENIX NSDI*, 2005.

[50] V. Kotronis, X. Dimitropoulos, and B. Ager, "Outsourcing the routing control logic: Better internet routing based on sdn principles," in *Proc. of Hotnets*. ACM, 2012, pp. 55–60.

[51] P. Traina, D. McPherson, and J. Scudder, "Autonomous system confederations for bgp," 2007. [Online]. Available: https://tools.ietf.org/html/rfc5065

[52] V. H. Dixit, S. Kyung, Z. Zhao, A. Doupé, Y. Shoshitaishvili, and G.-J. Ahn, "Challenges and preparedness of sdn-based firewalls," in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2018, pp. 33–38.

[53] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, "Brew: A security policy analysis framework for distributed sdn-based cloud environments," *IEEE transactions on dependable and secure computing*, vol. 16, no. 6, pp. 1011–1025, 2017.

[54] B.-H. Oh, S. Vural, N. Wang, and R. Tafazolli, "Priority-based flow control for dynamic and reliable flow management in sdn," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1720–1732, 2018.

[55] G. S. Aujla, R. Chaudhary, N. Kumar, R. Kumar, and J. J. Rodrigues, "An ensembled scheme for qos-aware traffic flow management in software defined networks," in *2018 IEEE international conference on communications (ICC)*. IEEE, 2018, pp. 1–7.

[56] B. Charyyev, E. Arslan, and M. H. Gunes, "Latency comparison of cloud datacenters and edge servers," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[57] B. L. Muhammad-Bello and M. Aritsugi, "TCloud: A transparent framework for public cloud service comparison," in *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, 2016, pp. 228–233.

[58] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *ACM Queue*, December 2013.

[59] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Communication Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[60] G. Grigoryan, Y. Liu, and M. Kwon, "PFCA: A programmable FIB caching architecture," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1872–1884, 2020.

[61] "The CAIDA UCSD Anonymized Internet Traces 2014," http://www.caida.org/data/passive/passive_2014_dataset.xml. [Online]. Available: http://www.caida.org/data/passive/passive_2014_dataset.xml

[62] Linux Foundation Collaborative Project, "FRRouting (FRR)," accessed: October, 2019. [Online] Available: https://frrouting.org.

[63] "RYU SDN Framework," https://osrg.github.io/ryu/.

[64] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5 – 23, 2014.

[65] GENI, "GENI Glossary," https://www.geni.net/documentation/glossary/.

[66] CAIDA, "The CAIDA UCSD Anonymized Internet Traces - 20181220," 2018, http://www.caida.org/data/passive/passive_dataset.xml.

[67] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

**Prasun Kanti Dey** (S'16) received his B.Sc. degree in Computer Science and Engineering (CSE) from Bangladesh University of Engineering and Technology, Bangladesh. He received his M.Sc. in CSE from University of Nevada - Reno, Reno, NV in 2016 and Ph.D. in Computer Engineering from University of Central Florida in 2019. He is currently with MathWorks Inc. as a Senior Software Engineer. His research interests include network management and security, SDN and distributed systems, network measurement and performance analysis, and network economics. He is a member of both ACM and IEEE.

**Murat Yuksel** (SM'11) is a Professor at the ECE Department of the University of Central Florida (UCF), Orlando, FL. He served as the Interim Chair of ECE at UCF from 2021 to 2022. Prior to UCF, he was a faculty member at the CSE Department of the University of Nevada, Reno, NV. From 2002 to 2006, he was a member of Adjunct Faculty and a Postdoctoral Associate at the ECSE Department of Rensselaer Polytechnic Institute (RPI), Troy, NY. He received his B.S. degree in computer engineering from Ege University, Izmir, Turkey in 1996, and M.S. and Ph.D. degrees in computer science from RPI in 1999 and 2002, respectively. He worked as a software engineer at Pepperdata, and a visiting researcher at AT&T Labs and Los Alamos National Lab. His research interests are in the areas of networked, wireless, and computer systems. He has been on the editorial boards of Computer Networks, IEEE Networking Letters, and IEEE Transactions on Machine Learning in Communications and Networking. He has published more than 200 papers at peer-reviewed journals and conferences, and is a co-recipient of three Best Paper, one Best Paper Runner-up, and one Best Demo Awards. He is a senior member of IEEE, and a senior and life member of ACM.