Egret: Simplifying Traffic Management for Physical and Virtual Network Functions

Yikai Lin
yklin@umich.edu
University of Michigan

Ajay Mahimkar
mahimkar@research.att.com
AT&T Labs Research

Bo Han
bohan@research.att.com
AT&T Labs Research

Zihui Ge
gezihui@research.att.com
AT&T Labs Research

Vijay Gopalakrishnan
gvijay@research.att.com
AT&T Labs Research

Z. Morley Mao
zmao@umich.edu
University of Michigan

ABSTRACT

Traffic migration is a common procedure performed by operators during planned maintenance and unexpected incidents to prevent/reduce service disruptions. However, current practices of traffic migration often couple operators' intentions (e.g. device upgrades) with network setups (e.g. load-balancers), resulting in poor re-usability and substantial operational complexities. Our study of 205 Methods of Procedure (MOPs) from a major U.S. carrier suggests that generalizing traffic migration with a unified model is feasible. Such generalization along with SDN's automation capability is key to scalable and flexible management of traffic, especially for virtualized network functions with unprecedented scale, heterogeneity, and fast iteration. In this paper, we propose *Egret*, a generic traffic migration system that simplifies traffic management for physical and virtual network functions. Egret (1) hides intricate implementation details from operators with generic intention-based interfaces, and (2) modularizes common traffic migration procedures to enable plug-and-play by developers and vendors. Leveraging a novel mask-based abstraction of traffic migration jobs, Egret can further simplify reverse traffic migration and enable job interleaving.

CCS CONCEPTS

• **Networks → Programmable networks**; **Network management**; *Programming interfaces*; *Network manageability*.

KEYWORDS

Network Functions Virtualization, Software Defined Networking, Traffic Management

ACM Reference Format:
Yikai Lin, Ajay Mahimkar, Bo Han, Zihui Ge, Vijay Gopalakrishnan, and Z. Morley Mao. 2019. Egret: Simplifying Traffic Management for Physical and Virtual Network Functions. In *The 15th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '19), December 9–12, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3359989.3365409

1 INTRODUCTION

Despite the advancement and growing adoption of Software Defined Networks (SDN), it remains challenging to perform traffic migration on a heterogeneous data plane, because different types of network functions have (or depend on) drastically different traffic steering capabilities/methods. For example, a router can easily drain its traffic by increasing OSPF weights of its links, but it is incapable of controlling what flows to drain or specifying destinations for the drained traffic; in contrast, a firewall often has to rely on external functions like load balancers or switches to steer traffic which do support more fine-grained controls.

These differences lead to highly customized solutions that tailor to specific applications (e.g. disaster mitigation [30]) and/or network setups (e.g. load-balancers and programmable switches). Be they traditional Methods of Procedure (MOPs) or systematic approaches [12, 21, 30], specialized solutions often require excessive knowledge of the infrastructure to develop and use (e.g. topology, device type). As network function virtualization becomes more prominent and network data plane more heterogeneous and dynamic, developing and using specialized solutions will inevitably become unsustainable and costly due to their poor reusability across applications and network functions.

We believe generalization is key to fundamentally addressing this problem because a unified and extensible model will greatly simplify (1) development of solutions by vendors (extension over customization) and (2) execution of solutions by operators (standardized interfaces over proprietary interfaces). To this end, we conduct a study on 205 change management MOPs (§2.1) from a major U.S. carrier to identify the commonalities and differences in traffic migration for different network functions in different scenarios. We find that:

(1) Four key parameters, *Target*, *Peer(s)*, *Weight(s)* and *Filter(s)* are sufficient to describe any traffic migration intentions. *Target*, around which traffic migration is performed, is the only required parameter (§2.3). (2) All traffic migration methods can be categorized as either local or remote. Local traffic migration takes place on the *Target* (e.g. a router steers traffic through OSPF weight adjustment), whereas remote traffic migration takes place on external network components such as load-balancers, DNS servers and programmable switches, which we refer to as *Anchor Points* (§2.4). (3) Major sources of complexities of traffic migration come from migration method & anchor point discovery, target & anchor point configuration, reverse traffic migration and parallel job coordination (§2.5).

CoNEXT '19, December 9–12, 2019, Orlando, FL, USA

ACM ISBN 978-1-4503-6998-5/19/12…$15.00
https://doi.org/10.1145/3359989.3365409

Based on these findings, we design *Egret*, a generic traffic migration system that simplifies traffic management for different network functions. Egret decouples specification of traffic migration intentions from intrinsic configuration details with its generic interfaces. It modularizes common stages of traffic migration workflows to enable vendors to plug-and-play. To further reduce the complexity of managing states of traffic migrations, Egret uses a mask-based abstraction to efficiently keep track of individual jobs to simplify reverse traffic migration and parallelizing jobs.

To summarize, we make the following contributions:

- We conduct the first comprehensive study of traffic migration as a general procedure through extensive analysis of 205 change management MOPs from a major U.S. carrier. We identify common patterns, stages and key parameters that help define a unified model of traffic migration. We also shed light on major sources of complexities in existing traffic migration practices.
- We propose the design of Egret, a traffic migration system that simplifies traffic management for different network functions for both operators and vendors through generalization, automation, and modularization.
- We prototype Egret using Berkeley Extensible Software Switch (BESS) and show that it supports existing Methods of Procedure while significantly simplifying traffic migration.

This paper is organized as follows: We present results and findings of our study on 205 change management MOPs in §2. Based on the findings, we propose the design of Egret in §3. We describe our prototype of Egret, demonstrate its capability through simulation and quantify its complexity reduction in §4.

2 MOP ANALYSIS

To better understand how different types of network functions perform traffic migration in practice, we perform an extensive analysis of 205 change management Methods of Procedure (MOPs) from a major U.S. carrier.

In this section, we first give an overview of the MOPs (§2.1), then we select three representative examples of those MOPs (§2.2) to help illustrate our findings (§2.3,§2.4,§2.5).

2.1 Method of Procedure

Methods of Procedure (MOPs) are manuals/documents that provide step-by-step instructions on how to perform an operation. Our study focuses on change management MOPs that describe the process of imposing changes such as software upgrades to specific network components. This process usually contains locking, health checks[1], state/configuration preservation, traffic migration, change deployment, reverse traffic migration, state/configuration restoration, and unlocking. In this work, we focus on the traffic migration procedure of these MOPs.

2.2 Representative MOP Examples

Next, we will show snippets (with omissions and translated into plain language) of 3 representative examples from the 205 MOPs.

2.2.1 PE Router Software Upgrade.

(1) **Increase the OSPF weights** on the PE[2] router to 65535
(2) Verify that traffic is drained
(3) Shutdown BGP sessions to CE routers
(4) Upgrade the drained PE router
(5) **Reset the OSPF weights** on the PE router
(6) Re-establish BGP sessions with CE routers

The above example shows how the operator drains traffic off a PE router before performing device upgrade, and "brings back" traffic after the change. This MOP is a perfect example of how minimal the input can be for a traffic migration job. Since 65535 is the maximum and default OSPF weight for drains, and software upgrade requires all links be drained, what operators need to provide as input for this traffic migration job is simply **which PE router**.

Note that in this example, to "bring back" traffic after the upgrade by resetting the OSPF weights, the operator needs to record the original OSPF weights. Additionally, this pattern of two opposite traffic migrations before and after a change is deployed is very common among the MOPs in our study. We will talk more about that in §2.5.

> **Observations. 1.** The input of a traffic migration job can be as minimal as the identifier of the target. **2.** It is common to perform two opposite traffic migrations during an operation, which usually requires the operator to maintain certain state variables throughout the process.

2.2.2 MME Software Upgrade.

(1) **Reduce "RelativeMmeCapacity"** of the target MME[3] to 0 so that incoming new connections will be redirected to other MMEs in the pool
(2) Verify that number of connected UEs is below threshold
(3) **Move connected UEs** to other MMEs in the pool
(4) Delete static routes towards the isolated MME
(5) Upgrade the isolated MME
(6) **Reset "RelativeMmeCapacity"** to its original value
(7) Reinstall the static routes towards the target MME

In the above example, the operator needs to isolate an MME from its pool before the upgrade, then insert the MME back into its pool. Unlike routers, MMEs are **stateful** — the operator needs to migrate existing and redirect new traffic in order to isolate the MME from its pool.

The approach to preventing new connections on an MME is conceptually similar to that of a PE router: one lowers the capacity while the other raises the weight, both leading to a lower preferability. However, to migrate existing traffic, the operator has to (1) query the number of connected UEs on the MME and (2) explicitly move these UEs to other MMEs.

> **Observations. 3.** Stateful network functions require two separate migrations for both new and existing traffic. **4.** Existing and new traffic are never distributed differently in any of the MOPs, and they are usually evenly distributed among peers.

[1] Health checks are performed after each step.

[2] Provider-Edge

[3] Mobility Management Entity, an LTE control-plane function

Table 1: Key Traffic Migration Parameters

Parameter	Usage	Required?	Percentage
Target	The network entity around which traffic migration is performed.	Yes	100%
Peer(s)	Network entities that receive or send traffic from or to the Target.	No	23%
Weight(s)	In each migration, the distribution of traffic among the receiving entities	No	22%
Filter(s)	Identifiers for any subset of the migrating traffic (e.g. bit-masks)	No	1%

2.2.3 *Network Zone Traffic Migration through DNS Redirection.*

(1) **Edit a list of DNS files** on DNS server "alnapnrdns01" to move traffic of Phone, Broadband, VoLTE, etc. to ALN[4] NZ[5]1, NZ3, NZ4, NZ5 and NZ6
(2) **Edit a list of DNS files** on DNS server "alnapnrdns01" to move primary roaming traffic to ALN NZ3, NZ4

In this example, the operator draining an entire network zone 2 by migrating traffic of specific services to other network zones via DNS redirection. This MOP differs from the previous two in that operators are using DNS servers to steer traffic as opposed to directly reconfiguring target entities. Besides, for this operation, the operator is very specific about where each service traffic should go; the destinations of different service traffic cannot be inferred.

Observations. 5. Network function is not the only type of target for a traffic migration job.
6. Traffic migration can happen on a more fine-grained level which involves a subset of the traffic of the target.

2.3 Key Parameters

We see from previous examples how simple traffic migration intentions get obscured by the detailed mechanics of specific procedures that vary drastically across network functions (NFs). To decouple the expression of intention and the actual execution details, we need to find a set of well-defined parameters that are generic yet carry enough information for specifics to be derived. Based on our analysis of the 205 MOPs, we identify four key parameters that satisfy this requirement, as shown in Table 1. To simply show their usage, here are how the three examples from §2.2 can be described using one or more of these parameters:

Traffic Migration for PE Router Software Upgrade.

(1) Drain the traffic off {Target="Router-1"}
(2) Bring back the traffic to {Target="Router-1"}

Traffic Migration for MME Software Upgrade.

(1) Send traffic of {Target="MME-1"} to
{Peers=["MME-2", "MME-3", ..., "MME-11"]} with
{Weights=[10%, 10%, ..., 10%]
(2) Bring back the traffic to {Target="MME-1"}

Network Zone Traffic Migration through DNS Redirection.

(1) Send traffic from {Target="NZ2"}
matching {Filters=["Phone","Broadband","VoLTE"]} to
{Peers=["NZ1","NZ3","NZ4","NZ5","NZ6"]}
(2) Send traffic from {Target="NZ2"}
matching {Filters=["Roaming"]} to
{Peers=["NZ3","NZ4"]}

2.4 Local and Remote Traffic Migrations

In previous examples, we can see that in some cases the operator performs traffic migration by directly reconfiguring the target; while in others, the operator uses an external component like a DNS server. We categorize these two methods as *local* and *remote* migrations, and call the external component an ***Anchor Point***. The difference between *local* and *remote* traffic migrations is whether it takes place on the *Target* or on *Anchor Points*.

2.5 Traffic Migration Complexities

As mentioned earlier, existing practices of traffic migration often couple operator's intention with network setups, which lead to complexities in the forms of additional information to acquire/maintain before/during each operation. In this section, we highlight these complexities identified in the MOP analysis.

Migration Method & Anchor Point Discovery For each traffic migration, operators need to determine what migration method the *Target* uses and identify the type and location of *Anchor Points* (if needed) based on their network setup (Figure 2).

Target or Anchor Point Configuration For both local and remote migrations, the diversity of *Targets* and *Anchor Points* translates unfortunately well to the diversity of their capabilities and how they are configured. *Targets* are not necessarily network functions, and stateful *Targets* incur additional procedures.

Reverse Traffic Migration As mentioned earlier, operators need to maintain state variables for reverse traffic migrations. Such externally maintained states not only incur additional operational complexities, but they also leave operators susceptible to run-time failures.

Parallel Job Coordination Our MOPs suggest that operators always perform operations sequentially even when there are multiple devices in the same pool that need to be upgraded. This results in maintenance windows spanning hours if not days, which is highly inefficient. Manually performing parallel operations is error-prone and complex, especially when the same anchor points are shared.

3 EGRET

From §2.3 we see that a few key parameters suffice to describe traffic migration intentions regardless of network function types and migration methods. This means that the amount of input required to carry out a traffic migration is minimal. Enlightened by this finding and observations on major traffic migration complexities (§2.5), we propose the design of Egret, a traffic migration system that (1)

[4] Acronym of a location
[5] Network zone

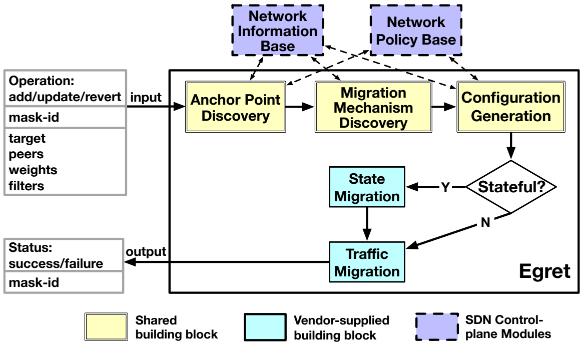

Figure 1: Egret's generic model and modular workflow

decouples high-level traffic migration intentions from low-level executions with a generic interface, and (2) modularizes low-level executions into functional building blocks that can be either shared or swapped in a plug-and-play fashion. In this section, we first present an overview of Egret's generic model and its modular workflow in §3.1, then we introduce an optimization Egret leverages to efficiently manage existing and in-coming traffic migration jobs in §3.2.

3.1 A Unified Model and Building Blocks

Figure 1 shows an overview of Egret's design. Egret allows operators to specify their traffic migration intentions through a generic interface based on the key parameters described earlier: *Target*, *Peer(s)*, *Weight(s)* and *Filter(s)*. Some of the procedures discussed in §2.5, which contribute to most of the complexities of MOPs and now abstracted by Egret's interface, are modularized as building blocks in Egret's internal workflow (detailed below). Some of these building blocks are reusable, such as anchor point discovery and migration mechanism discovery, because they are universal and fundamental to all MOPs according to our study. Others are specific to network function types or anchor point types, such as state migration and traffic migration since they directly interface with underlying network functions. Vendors only need to develop these two building blocks in Egret's modular design, as opposed to writing completely new MOPs from scratch.

Anchor Point Discovery. As discussed in §2.5, traffic migration intentions are independent of the type and locations of anchor points. However, as shown by previous works [16, 21], the selection of anchor points does play an important role in traffic migrations in terms of performance(e.g. packet loss) and service impact. Egret's anchor point discovery building block leverages network topology information stored in a network information base (NIB, commonly found in SDN control platforms) and follows operators' policy to find the most suitable anchor point(s) (if necessary). Note that there can be multiple anchor points for one migration depending on the relative positions of the *Target* and *Peers(s)* in the network. Figure 2 shows a simple example where the servers' migration method is *remote* and their anchor points are switches.

Migration Mechanism Discovery. In §2.4 traffic migration methods are categorized as either *local* or *remote* based on where migration happens. After the discovery of *anchor points* from the topology, this building block determines the actual migration method the *Target* uses by querying the NIB.

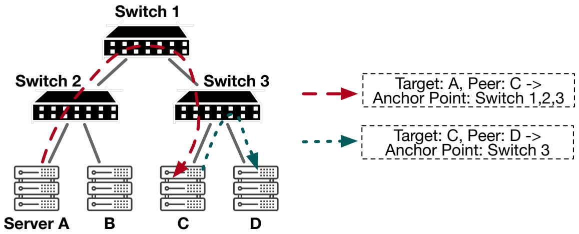
Switch 1
Switch 2
Switch 3
Server A
B
C
D
Target: A, Peer: C -> Anchor Point: Switch 1,2,3
Target: C, Peer: D -> Anchor Point: Switch 3

Figure 2: Anchor point discovery in a simple tree topology

Configuration Generation. This building block takes in parameters from input plus *anchor point* and migration method information from previous building blocks, and generates actual configurations for the corresponding network components. These configurations can be as simple as a single command to raise OSPF weight to 65535 on a router link or as complex as many flow table entries for multiple programmable switches [21]. In this building block, a mask-like abstraction is used for efficient management of existing and in-coming traffic migration jobs (detailed in §3.2).

Traffic & State Migration. These two building blocks directly interface with *Targets* or *Anchor Points* to deploy the configuration changes for actual migrations. Compared with MOPs which are application-, software version- and network setup-specific, these two building blocks are, in the worst case, *Target*- or *Anchor Point*-specific. Leveraging generic configuration solutions like OpenConfig [4], these two building blocks can be further generalized and vendors' effort reduced.

3.2 Mask-based Job Management

In §2.5 we mention that reverse traffic migration is one of the major sources of complexities in existing traffic migration workflows because it requires operators to (1) keep track of deployed configurations and (2) compute new configurations to revert existing ones. To minimize the amount of state operators have to maintain and enable easy reversions, we propose the design of a mask-based abstraction for traffic migration jobs that is easy to manage (add, update, revert). This design is inspired by configuration rollback capabilities commonly found in modern routers [8]. Egret's mask-based APIs are shown in Table 2.

Figure 3 shows an example of a 3-port load-balancer. Each port has a weight, indicating what portion of in-coming traffic should go out through each port. A mask (corresponding to one traffic migration request) consists of relative weights indicating the amount of traffic to be taken from or given to each port in a migration. Masks are put on top of one another into a stack, at the bottom of which is the *base* denoting the default configuration of the load balancer. The current (deployed) distribution is calculated by applying all masks to the *base*. In this example, the *base* is {30%,30%,40%}, indicating by default 30% of traffic goes out of port 1 and so on. In Figure 3-b a new traffic migration job comes in with the intention to migrate traffic from *Target* 1 to 2 and 3 (assuming *Targets* 1, 2 and 3 are connected to port 1, 2 and 3 respectively). This will generate a new mask {-100%,+50%,+50%} on top of the *base* resulting in a new configuration {0%,45%,55%}[6]. New masks always go to the top of

[6] Because the 30% traffic of port 1 is distributed equally to port 2 and 3, each getting 15% of all in-coming traffic.

Table 2: Egret's mask-based APIs

API	Description
add([params])	add a new mask described by [params] need to be explicitly removed if timeout not specified
update(mask-id, [params])	augment the [params] of an existing mask associated with the mask-id, including the *base*
revert(mask-id)	remove the mask associated with the mask-id

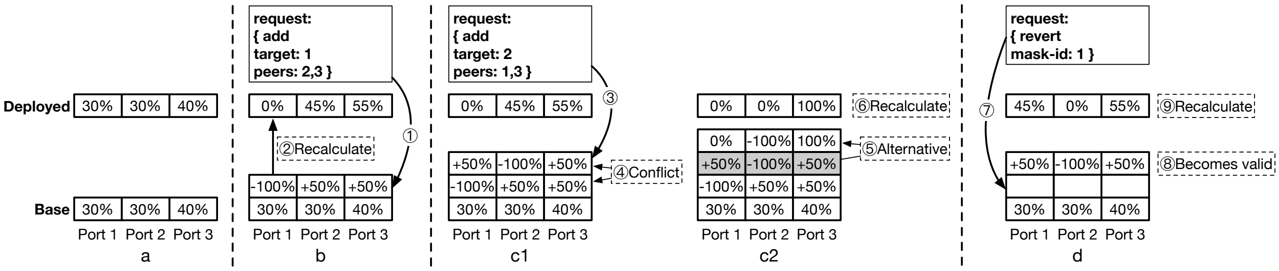

Figure 3: Managing traffic migration jobs with mask-like abstractions: a 3-port load-balancer example.

the stack, while existing masks can be removed from anywhere. Whenever such a change happens, the current configuration is recalculated automatically.

3.2.1 Job interleaving and conflict resolution. Through mask stacking, Egret allows different traffic migration jobs to interleave, meaning individual jobs can start and finish independently (assuming no conflicts exist between them). This is a significant improvement over traditional MOPs which are run sequentially (§2.5) and it is non-trivial to parallelize jobs when anchor points are shared. Egret achieves job interleaving by keeping track of individual masks and always recalculating actual configuration whenever changes happen. For example, in Figure 3-c1, a second job intending to migrate traffic from *Target* 2 to 1 and 3 arrives while job 1 is still active. The two jobs conflict with each other because *Target* 1 cannot be both source and sink. However, *Target* 3 is a sink in both requests, meaning an alternative exists for job 2. Policy-permitted, Egret can automatically resolve conflicts and generate alternative masks such as the one in Figure 3-c2. Additionally, in Figure 3-d, job 1 is reverted while job 2 is active. Since Egret always recalculates the configuration upon changes, the original mask 2 becomes valid and is applied to *base*. Job 1 and job 2 are never "aware" of each other throughout this process.

3.2.2 Achieving reverse traffic migration. In Figure 1, we see that Egret returns a mask-id with each execution. This allows operators to uniquely identify existing jobs. As mentioned in §2.5, reverse traffic migration brings an enormous amount of complexity because additional states need to be maintained and new configurations need to be generated. In Egret, with each mask kept track of individually, reverse traffic migration becomes as simple as reverting a mask, as shown in Figure 3-d.

3.2.3 Updating the base and handling runtime failures. Sometimes the operator needs to re-adjust the default configuration of a network component, or more often a failure happens rendering the existing configuration obsolete (e.g. the number of available ports decreases because of a failed link). These situations can be easily handled by updating the *base*, either proactively by the operator or automatically through common SDN topology discovery mechanisms [2, 5, 14, 22].

4 EVALUATION

In this section, we briefly describe our prototype of Egret using Berkeley Extensible Software Switch (BESS) [1] and use simulation to show how Egret handles failures during traffic migration. We evaluate how effective Egret is in simplifying traffic migration by measuring the operations to perform and information to manage by operators with and without Egret.

4.1 Prototype of Egret

We implement Egret's building blocks (§3.1) in Python that communicate with BESS simulated data-planes through gRPC [13]. Our prototype is capable of migrating traffic across multiple anchor points with a simple algorithm that finds parent nodes of *Target* and *Peer(s)* and their closest common ancestor. This allows Egret to migrate traffic between servers on different racks which involve reconfiguring ToR switches and aggregate switches (See Figure 2). Yet the input to trigger that migration is as **simple** as {Target: server-A, Peer: server-C}.

4.1.1 Handling run-time failures. To demonstrate Egret's mask-based job management, especially how it handling run-time failures by automatically updating the *base* (§3.2), we use BESS to simulate an L4 load-balancer connecting to three firewalls. In this experiment, we manually disconnect firewall-2 from the load-balancer to emulate a failure after submitting a traffic migration request to migrate traffic away from firewall-1 through Egret's API and observe how traffic distribution changes throughout the entire process. The results are shown in Figure 4. The input in this case is even **simpler** than the last: {Target: firewall-1}.

4.2 Traffic Migration Simplification

By generalizing traffic migration with its model and decoupling intention specification from executions for automation, Egret can significantly reduce the complexities for traffic migration compared to traditional MOPs, as shown in Table 3. One reduction is in the

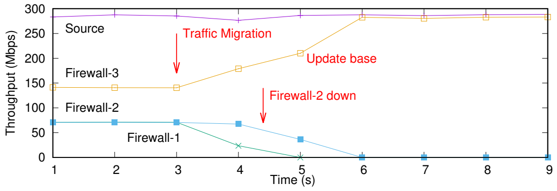

Figure 4: Egret handling run-time failure.

number of operations performed by the operator. As mentioned earlier, traditional MOPs require operators to run low-level configuration commands or modify configuration files line by line (e.g. §2.2.3), which is error-prone and time-consuming, especially when the number of operations increases with the size of the network. Egret reduces this complexity by allowing operators to express their intentions using the mask-based API with a few key parameters. As shown in Table 3, Egret consumes only two API calls for jobs that require reverse traffic migration (add and revert mask respectively). Another significant reduction in complexity is the number of state variables that need to be acquired and maintained during traffic migrations. Using Egret, operators only need to know the *Target* in the router example; *Target*, *Peers* and *Weights* in the MME example; and *Target*, *Peers* and *Filters* in the APN-DNS example. And the only state that is maintained is mask-id. Whereas in MOPs, operators need to record existing configurations and states (link OSPF weights, MME connection stats, DNS entries, etc.) while running tens or hundreds of lines of commands.

5 DISCUSSION

Egret's applicability to all traffic migration applications and network function types. Even though Egret applies to the 205 MOPs from our study, it is impossible to claim universal applicability. We do, however, make a few design choices to future-proof Egret: (1) the *Filter* parameter of Egret's API supports any packet matching patterns in the form of bit-masks; (2) Egret's categorization of migration methods is very coarse-grained: *local* and *remote*, which makes it easy for vendors to plug-and-play building blocks.

Conflict detection with Egret's mask-based approach. As mentioned earlier (§3.2), Egret rejects jobs that use existing traffic sources as sinks and vice versa. This is intuitive for jobs that intend to completely drain their *Targets*. For jobs that migrate traffic partially, however, this policy can be too strict. Exploring more flexible policies would be an interesting next step.

Time reduction by Egret. With automation and job parallelization, there should be a considerable amount of time reduction by Egret compared to MOPs. It is, however, non-trivial to quantify this reduction on traffic migration alone since it is among many other procedures in a MOP. We plan to further work with the operation teams and address this problem in our future work.

6 RELATED WORK

Traffic management, especially in the context of SDN, is not a new topic. Previous works have provided many instrumental tools — scalable and efficient rule management in control and data-plane [20, 25, 26], abstractions for forwarding elements [7, 17, 19], network function state management [12, 29, 31], and traffic engineering [15, 23]. Egret's ultimate goal is to be able to incorporate and leverage all these tools for different kinds of applications in different network setups, while maintaining a unified interface.

Table 3: Comparison in amount of operations to execute and information to acquire/maintain by the operator

	# of operations		# of state variables	
	MOP	Egret	MOP	Egret (+1 for mask-id)
Router	5-10	2	# of links	1+1
MME	>40	2	>10	3+1
DNS	>100	1	>50	3+1

Maelstrom [30] is a traffic management framework that shares some insights with Egret such as abstracting traffic migration intricacies with generic interfaces and providing reusable primitives to compose workflows. However, Maelstrom is tailored to traffic management for disaster mitigation and recovery and is based on a load-balancer-only setup. Egret is a more general approach that abstracts traffic migration from different applications and network setups. Particularly, Egret addresses the heterogeneity of network functions, which is less prominent in data-center settings such as maelstrom's. Maelstrom has also addressed several technical challenges for traffic migration such as preserving service dependencies, which can be incorporated in Egret's building blocks.

Load balancing is a common way of traffic migration. Previous works either use software load-balancers [3, 9–11, 28] or propose new solutions [6, 24, 27] for traffic shifting. While these works focus on improving load-balancing, Egret focuses on abstracting traffic migration, with load-balancing being one of the *remote* migration methods.

Network function state management is closely related to traffic migration. OpenNF [12], S6 [31] and OFM [29] all use an SDN-based mechanism (SDN switch) to steer traffic which is another way of *remote* migration in Egret's model. Khalid et al. [18] proposed standardized APIs for VNF management but didn't focus on generalizing and simplifying traffic migration as Egret does.

7 CONCLUSION

In this paper, we revisit traffic migration, a common procedure in many network operations, in the light of rapidly expanding virtualized network functions. Our analysis of 205 change management MOPs sheds light on the commonalities and complexities of existing traffic migration practices which motivate the design of Egret. Compared to traditional MOPs, Egret can significantly simplify traffic migration through generalization (a unified model and generic API), automation, and efficient state management (mask-based abstraction).

ACKNOWLEDGMENTS

We would like to thank our shepherd, Anja Feldmann, and the anonymous reviewers for their constructive and valuable feedback. We are also grateful to Jennifer Yates and Marco Platania at AT&T Labs Research for their input on this work. This work was partially supported by NSF under awards CCF-1628991 and CNS-1544678.

REFERENCES

[1] 2019. BESS: Berkeley Extensible Software Switch. https://github.com/NetSys/bess.

[2] 2019. Floodlight OpenFlow Controller. http://www.projectfloodlight.org/floodlight/.

[3] 2019. Istio / Traffic Shifting. https://istio.io/docs/tasks/traffic-management/traffic-shifting/.

[4] 2019. OpenConfig. http://openconfig.net/.

[5] 2019. Ryu SDN Framework. https://osrg.github.io/ryu/.

[6] João Taveira Araújo, Lorenzo Saino, Lennert Buytenhek, and Raul Landa. 2018. Balancing on the edge: Transport affinity without network state. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), Renton, WA*.

[7] Martin Casado, Teemu Koponen, Rajiv Ramanathan, and Scott Shenker. 2010. Virtualizing the network forwarding plane. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*. ACM, 8.

[8] Cisco. 2019. Managing Configuration Files Configuration Guide, Cisco IOS XE Release 3S - Configuration Replace and Configuration Rollback [Cisco IOS XE 3S] - Cisco. https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/config-mgmt/configuration/xe-3s/config-mgmt-xe-3s-book/cm-config-rollback.html.

[9] Daniel E Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A Fast and Reliable Software Network Load Balancer.. In *NSDI*. 523–535.

[10] Rohan Gandhi, Y Charlie Hu, Cheng-Kok Koh, Hongqiang Harry Liu, and Ming Zhang. 2015. Rubik: unlocking the power of locality and end-point flexibility in cloud scale load balancing. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*. 473–485.

[11] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. 2015. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 27–38.

[12] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. 2014. OpenNF: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 163–174.

[13] Google. 2019. gRPC. https://grpc.io/.

[14] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. 2008. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review* 38, 3 (2008), 105–110.

[15] David Ke Hong, Yadi Ma, Sujata Banerjee, and Z Morley Mao. 2016. Incremental deployment of SDN in hybrid enterprise and ISP networks. In *Proceedings of the Symposium on SDN Research*. ACM, 1.

[16] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. 2014. Dynamic scheduling of network updates. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 539–550.

[17] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. 2013. Optimizing the one big switch abstraction in software-defined networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 13–24.

[18] Junaid Khalid, Mark Coatsworth, Aaron Gember-Jacobson, and Aditya Akella. 2016. A standardized southbound API for VNF management. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*. ACM, 38–43.

[19] Yikai Lin, Ulaş C Kozat, John Kaippallimalil, Mehrdad Moradi, Anthony CK Soong, and Z Morley Mao. 2018. Pausing and resuming network flows using programmable buffers. In *Proceedings of the Symposium on SDN Research*. ACM, 7.

[20] Alex X Liu, Chad R Meiners, and Eric Torng. 2010. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. *IEEE/ACM Transactions on Networking (TON)* 18, 2 (2010), 490–500.

[21] Hongqiang Harry Liu, Xin Wu, Ming Zhang, Lihua Yuan, Roger Wattenhofer, and David Maltz. 2013. zUpdate: Updating data center networks with zero loss. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 411–422.

[22] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. 2014. Opendaylight: Towards a model-driven sdn controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 1–6.

[23] Alaitz Mendiola, Jasone Astorga, Eduardo Jacob, and Marivi Higuero. 2016. A survey on the contributions of software-defined networking to traffic engineering. *IEEE Communications Surveys & Tutorials* 19, 2 (2016), 918–953.

[24] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 15–28.

[25] Masoud Moshref, Minlan Yu, Abhishek Sharma, and Ramesh Govindan. 2012. vcrib: Virtualized rule management in the cloud. In *Presented as part of the*.

[26] Masoud Moshref, Minlan Yu, Abhishek Sharma, and Ramesh Govindan. 2013. Scalable rule management for data centers. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*. 157–170.

[27] Vladimir Olteanu, Alexandru Agache, Andrei Voinescu, and Costin Raiciu. 2018. Stateless datacenter load-balancing with beamer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, Vol. 18. 125–139.

[28] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A Maltz, Randy Kern, Hemant Kumar, Marios Zikos, Hongyu Wu, et al. 2013. Ananta: Cloud scale load balancing. In *ACM SIGCOMM Computer Communication Review*, Vol. 43. ACM, 207–218.

[29] Chen Sun, Jun Bi, Zili Meng, Xiao Zhang, and Hongxin Hu. 2018. OFM: Optimized flow migration for NFV elasticity control. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[30] Kaushik Veeraraghavan, Justin Meza, Scott Michelson, Sankaralingam Panneerselvam, Alex Gyori, David Chou, Sonia Margulis, Daniel Obenshain, Shruti Padmanabha, Ashish Shah, et al. 2018. Maelstrom: mitigating datacenter-level disasters by draining interdependent traffic safely and efficiently. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 373–389.

[31] Shinae Woo, Justine Sherry, Sangjin Han, Sue Moon, Sylvia Ratnasamy, and Scott Shenker. 2018. Elastic Scaling of Stateful Network Functions. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*.