Dependable Virtualized Fabric on Programmable Data Plane

Kaihui Gao, Shuai Wang, Kun Qian, Dan Li, *Senior Member, IEEE*, Rui Miao, Bo Li, Yu Zhou, Ennan Zhai, Chen Sun, Jiaqi Gao, Dai Zhang, Binzhang Fu, Frank Kelly, Dennis Cai, Hongqiang Harry Liu, Yan Li, Hongwei Yang, and Tao Sun

***Abstract*— In modern multi-tenant data centers, each tenant desires reassuring dependability from the virtualized network fabric – *bandwidth guarantee with work conservation*, *bounded tail latency* and *resilient reachability*. However, the slow convergence of prior works under network dynamics and uncertainties can hardly provide the dependability for tenants. Further, state-of-the-art load balance schemes are guarantee-agnostic and bring great risks on breaking bandwidth guarantee, which is overlooked in prior works. In this paper, we propose vFab, a dependable virtualized fabric framework which can (1) quickly detect network failure in data plane, (2) explicitly select proper paths for all flows, and (3) converge to ideal bandwidth allocation at sub-millisecond. The core idea of vFab is to leverage the programmable data plane to build a fusion of an active edge (*e.g.*, NIC) and an informative core (*e.g.*, switch), where the core sends link status and tenant information to the edge via telemetry to help the latter make a timely and accurate decision on path selection and traffic admission. We fully implement vFab with commodity SmartNICs and programmable switches. Extensive evaluations show that vFab can keep bandwidth guarantee with high bandwidth utilization, low and bounded latency, and resilient reachability under various network scenarios with limited overhead. Application-level experiments show that vFab can improve QPS by 2.4× and cut tail latency by 10× compared to the alternatives.**

***Index Terms*— Data center network, programmable network, performance guarantee.**

Manuscript received 22 June 2022; revised 25 September 2022; accepted 15 November 2022; approved by IEEE/ACM *Transactions on Networking* Editor D. Han. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2021B0101400001, in part by the National Natural Science Foundation of China under Grant U21B2022, in part by the Tsinghua University-China Mobile Communications Group Corporation Ltd. Joint Institute, in part by the Tsinghua University (Department of Computer Science and Technology)-Siemens Ltd., and in part by the China Joint Research Center for Industrial Intelligence and Internet of Things. In this journal article, the authors added a new goal on resilient reachability, and the corresponding background, solution, implementation, and evaluation. Part of this work was published in ACM SIGCOMM 2022 [DOI: 10.1145/3544216.3544241]. *(Kaihui Gao and Shuai Wang contributed equally to this work.) (Corresponding author: Dan Li.)*

Kaihui Gao and Dan Li are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100190, China (e-mail: kaihuigao@163.com; tolidan@tsinghua.edu.cn).

Shuai Wang is with the Zhongguancun Laboratory, Beijing 100080, China.

Kun Qian, Rui Miao, Bo Li, Yu Zhou, Ennan Zhai, Chen Sun, Jiaqi Gao, Dai Zhang, Binzhang Fu, Dennis Cai, and Hongqiang Harry Liu are with Alibaba Group, Hangzhou 311121, China.

Frank Kelly is with the Faculty of Mathematics, University of Cambridge, CB2 1TN Cambridge, U.K.

Yan Li is with Siemens Ltd., Beijing 100102, China.

Hongwei Yang and Tao Sun are with China Mobile Research Institute, Beijing 100053, China.

Digital Object Identifier 10.1109/TNET.2022.3224617

I. Introduction

IN MODERN multi-tenant data centers, virtual machines (VMs) of a tenant are expected to be logically interconnected by a virtualized fabric (VF) as if in a dedicated cluster, even though all tenants share the same physical network. It is important to ensure the dependability of VFs – *bandwidth guarantee with work conservation*, *bounded tail latency* and *resilient reachability*. While many solutions have been proposed to improve the performance of multi-tenant DCNs (see Table I), they are not competent to provide a dependable VF for two reasons.

Firstly, the convergence speed ($>$ tens of milliseconds) of prior works fails to catch up with the increasingly rigorous performance demand from today's applications. Multiple factors drive this trend. First, resource pooling is an inevitable trend in data centers, which has strict resource access deadlines [2], [3], [4], [5]. For instance, the I/O latency of the enhanced SSD, which is the highest performance level disk in Elastic Block Storage, requires 200μs on average and $1ms$ at tails [6]. Second, distributed computation requires instantaneously available bandwidth every time the parameter/activation transfer starts, especially for AI inference, which typically involves multiple transfers and needs to respond to online queries within $10ms$ [7], [8]. Finally, the control plane (*e.g.*, BGP, SDN) can only handle certain types of failures (*e.g.*, link down) but gray failures (*e.g.*, random packet drops [9], [10]), and its recovery time would vary from seconds to minutes [11], [12], [13]. Long periods of loss of reachability can crash TCP throughput [14]. Hence, handling network dynamics and uncertainties rapidly, *i.e.*, at sub-millisecond timescales, is critical to meet the dependability requirements of applications.

Secondly, bandwidth guarantee could be easily broken by guarantee-agnostic load balancing schemes. Existing solutions [15], [16], [17], [18] providing bandwidth guarantee with work conservation mostly abstract the network fabric as an aggregated pipe between source and destination, and assume that specific path selections are done with orthogonal load balancing schemes, *e.g.*, selecting a random path [19] or the least-utilized path [20], [21], [22]. However, due to work conservation, *utilization* and *bandwidth subscription* in a link are not equivalent, *i.e.*, flows with low bandwidth guarantees may create high loads in a path (§ II-B), or vice versa. Note that guaranteeing the minimum bandwidth demands is mandatory, but providing extra capacity is a bonus. Hence, subscription-aware path selection is important to provide a dependable VF.

TABLE I

RELATED SYSTEMS. (*THE WORK CANNOT ISOLATE NETWORK PERFORMANCE; †THE WORK RESERVE BANDWIDTH OR USE NETWORK CALCULUS)

Systems	Resilient Reachability	Strict Bandwidth Isolation			Low Latency	Topology Requirement	Network Requirement
		Tenant-level Guarantee	Work Conservation	Convegence Speed			
FRR [23], F10 [24], DDC [25], ShareBackup [26]	✓	✗	✗	–*	✗	None	None
MPTCP [27], MPQUIC [28]	✓	✗	✓	–*	✗	None	None
QJUMP [29], Chameleon [30]	✗	✗	✗	–*	✓	None	Priority Queues
SecondNet [31], Oktopus [32], CloudMirror [33], Silo [34]	✗	✓	✗	–†	✓	None	None
Seawall [35], FairCloud [18] (PS-L/N)	✗	✗	✓	10∼200ms	✗	None	ECN
ElasticSwitch [15]	✗	✓	✓	10∼200ms	✗	None	ECN
NetShare [36]	✗	✓	✓	10∼200ms	✗	None	Per-Tenant Queue
FairCloud [18] (PS-P)	✗	✓	✓	10∼200ms	✗	Tree topology	Per-Tenant Queue
Hadrian [37]	✗	✓	✓	5∼10ms	✗	Tree topology	Programmable
Proteus [38], EyeQ [16], HUG [39], GateKeeper [40]	✗	✓	✓	5∼10ms	✗	congestion/loss-free fabric	None
PicNIC [17]	✗	✓	Partial	5∼10ms	Host side	congestion/loss-free fabric	None
vFab (Our work)	✓	✓	✓	<1ms	✓	None	Programmable

Fundamentally, we find that to make a VF converge more rapidly and select path more accurately, fine-grained network status, *e.g.*, link health, bandwidth subscription, queuing size, and utilization, is the key clue. However, due to prior works' inability to obtain fine-grained network status, they have to suffer from heuristic rate control and utilization-oriented load balancing. Fortunately, the emerging programmable switches and NICs are bringing possibilities to obtain real-time and precise network information, opening up new opportunities to build a dedicated dependable VF framework.

In this paper, we propose vFab, a framework which provides a strongly dependable VF service to data center tenants. By leveraging programmable switches and NICs in modern data centers, vFab simultaneously provides bandwidth guarantee with work conservation, bounded tail latency and resilient reachability in an end-to-end way. Specifically, vFab builds a fusion of an informative core (*e.g.*, switches) and an active edge (*e.g.*, NICs). In the core, each switch sends critical information, *e.g.*, link status and active bandwidth subscription, back to the edge via telemetry. With the real-time feedback from the core, the edge can achieve the expected network performance via rapid and accurate path selection and rate control. The informative data plane enables rapid detection and mitigation of performance degradation.

There are three challenges to realizing vFab. First, since data center traffic mix changes swiftly, it is necessary to quickly provision bandwidth once a tenant has traffic demand, while it is more efficient to allow other tenants to share the reserved but unused bandwidth. Second, data center traffic is bursty in nature, *e.g.*, incast, which requires the distributed framework to cooperatively admit traffic across hosts to avoid traffic interference among tenants at short timescales. Third, detecting path quality and migrating path are not easy, which may result in a prolonged convergence process and traffic oscillation.

vFab addresses these challenges with three innovations:

(i) *Hierarchical bandwidth allocation.* First, the edge selects a path for each flow to keep that the active bandwidth subscription, *i.e.*, the total minimum bandwidth demands of tenants that pass through the path, does not exceed the path capacity. Thus, the minimum bandwidth can be guaranteed for all tenants by proportional sharing. Then, the edge swiftly and accurately adjusts sending rate to make the link utilization converge to the target. Our theoretical analysis suggests that vFab can achieve both strict bandwidth guarantee and high network utilization (§ III-C);

(ii) *Two-stage and window-based traffic admission.* In order to avoid queuing in switch, each edge uses a window updated by real-time link utilization, *i.e.*, utilization-based window, to limit a tenant's inflight traffic on a path. Across hosts, vFab controls each tenant's total burst up to their bandwidth demand and additively increases their sending windows until utilization-based windows ramp down, and starts to use the latter. Thus, vFab can bound the queuing size on bottleneck link to three times of BDP (Bandwidth-Delay Product) (§ III-D);

(iii) *Failure detection and path migration.* The edge makes a timely and accurate judgment on available bandwidth and reachability on a path with a single probe. vFab designs a link failure self-detection mechanism (§ III-E) in the core. Once a link failure occurs (including gray failures), the upstream switch will detect it in time, and rebound the probe to notify the failure. Hence, the edge can swiftly select a proper path to migrate to for maintaining end-to-end performance without a lengthy convergence process or impacting other innocent tenants. vFab's path migration can also avoid oscillations and packet re-ordering (§ III-F).

We implement vFab in commodity SmartNICs and programmable switches. Experiments show that vFab can keep VF dependable under various network conditions, even under failures. Compared to the alternatives, vFab reduces the minimum bandwidth violation from >40% to nearly 0%, improves application-level QPS by 2.4×, and reduces 10× tail latency. Also, vFab converges 183× faster than the alternatives. Meanwhile, vFab can support tens of thousands of VM-pairs with $< 20\%$ extra hardware resources and $< 1.28\%$ probing bandwidth overhead. NS3 simulations verify that vFab can keep sub-millisecond convergence speed under real workload in large-scale topology (512 servers).

II. BACKGROUND AND MOTIVATION

A. Why Is Ensuring VF Dependability Hard?

In today's multi-tenant DCNs, ensuring VF dependability faces practical challenges for three reasons.

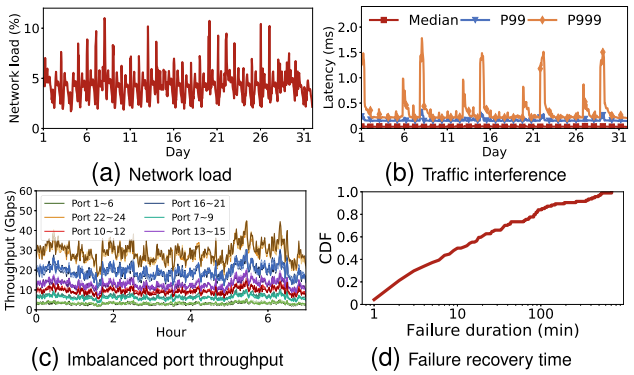


Fig. 1. Network uncertainties from a global cloud provider.

First, the traffic bursts in DCNs shared by uncooperative tenants are changing more and more dramatically, which requires ensuring VF dependability for tenants even at short timescales. Although the cloud provider typically overprovisions the bandwidth capacity, it cannot efficiently isolate network resources among tenants to accommodate the applications' increasingly sensitive demands at short timescales. For example, Figure 1b shows the round-trip time (RTT) of a tenant's traffic from a large cloud provider over one month and hourly-averaged network utilization in that cluster. While the hourly-averaged network utilization is below 10% (Figure 1a), the tenant observes a periodic bandwidth decline and up to 50× latency inflation in 99.9th percentile (P999) than the median, due to the traffic interference from another tenant (not shown) running a routine data analysis.

Second, although DCN fabric provides many equivalent paths for end-to-end transmission, load imbalance among multi-paths and congestion often occur. Figure 1c shows the port throughput of all 24 upstream links from an Aggregation switch. These upstream links are connected to different Core switches and therefore equivalent in transferring data from the current Pod to other Pods. However, the load of these links converges to 6 different levels. Link $22 \sim 24$ bears $10\times$ larger load than link $1 \sim 6$. Since ECMP is the de-facto mechanism for balancing load, the root cause is that both ToR switches and Aggregation switches use the same type of switch chip, which causes hash polarization [41]. Load imbalance will lead to low network utilization and VF bandwidth demands that cannot be guaranteed.

Third, DCNs are faced with various and prevalent network failures that are time-consuming to repair, making it necessary for the dependable VF to provide resilient reachability. Fail-stop failures, *e.g.*, link/device down, cause transient routing black holes that last for seconds or minutes for the control plane to converge again. Furthermore, gray failures, *e.g.*, random packet drops [9], [10], resort to the management plane to mitigate [42], [43], which is even slower. Figure 1d shows the network repair time over all incident tickets (> 200) for network failures amassed over the last two years in the same cloud provider, 86% of which are reachability issues caused by packet loss. With a median time-to-repair of 10 minutes, this is much slower than the requirement of dependable VF, *e.g.*, recovering at sub-millisecond.

B. State-of-the-Art Solutions Cannot Help Out

While there is a gap between production-deployed solutions and state-of-the-art ones, we argue that they all use heuristic evolution based on limited network information. This root cause determines that prior solutions (and their combinations) cannot accurately and efficiently converge to the desired network state of tenants. For example, PicNIC [17], the state-of-the-art scheme designed for predictable virtualized NIC, provides guaranteed performance at edges, but cannot address fabric congestion; Seawall [35], a representative of runtime network-wide bandwidth allocation schemes, uses weighted congestion control (WCC) algorithms to share network bandwidth proportionally but converges slowly (tens of milliseconds). WCC is widely deployed in typical bandwidth allocation schemes [15], [16], [35], [44], which makes them all fail to react to sub-millisecond-level traffic bursts. As a representative of per-flowlet load balancing, Clove [22] takes a step forward in path selection based on explicit path utilization, but still cannot provide a bandwidth guarantee due to the lack of tenant-level demand information. These drawbacks can significantly impact the dependability of VF performance. Next, we use experiments to illustrate this point.

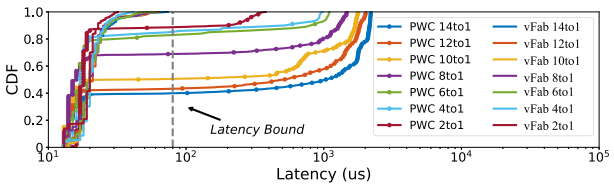


Fig. 2. RTT under various incast degrees. (Case-1).

Experiment settings : Our testbed has link capacity of 10Gbps, with the maximum base RTT ($baseRTT$) of 24μs. Like existing work [15], [16], [45], we set the target bandwidth utilization as 95%. Since we mainly focus on network resources, we only compare to PicNIC's components for bandwidth envelope, *i.e.*, weighted fair queues and receiver-driven CC, and we call it PicNIC′. This is similar to EyeQ [16]. We choose Swift [46], a latency-based CC recently proposed for DCN, as the basis of WCC, due to its excellent low latency. Clove [22] is deployed as the load balance mechanism, which balances traffic at the flowlet granularity and uses path utilization as the path selection metric.

Case-1: Greedy rate evolution may cause unbounded latency. We use an "incast" scenario to show that the combination of prior solutions (PicNIC′ +WCC+Clove, PWC) cannot guarantee bounded latency. N flows belonging to different VFs have the same destination host, and their minimum bandwidth demands are all $500Mbps$. They start to transmit traffic at the same time. N increase from 2 to 14, and Figure 2 shows the distribution of RTT under different incast degrees. The tail latency is positively associated with the incast degree. The root cause is that existing congestion control heuristically evolves sending rate to achieve full network utilization. Therefore, more active flows in the network would exaggerate the upper bound of the burst and finally lead to unbounded queuing latency.

Case-2: Slow convergence may break bandwidth guarantee. In Figure 3a, $F1$ is a long flow, and $F2$ periodically generates a $10MB$ message. Since $F1$ fully utilizes the capacity when $F2$ is absent (for work conservation), and $F2$ has to fetch its bandwidth back each time by competing for the bandwidth. Figure 3b shows that $F1$ and $F2$ cannot proportionally share bandwidth of the bottleneck link only with PicNIC′, while converging quickly due to the limitation of the maximum sending windows. Combining with WCC

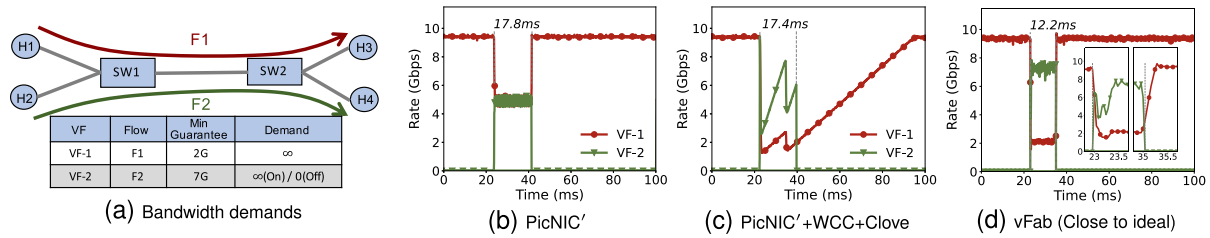


Fig. 3. Impractical assumption (b) or slow convergence (c) cannot consistently guarantee bandwidth demands. (Case-2).

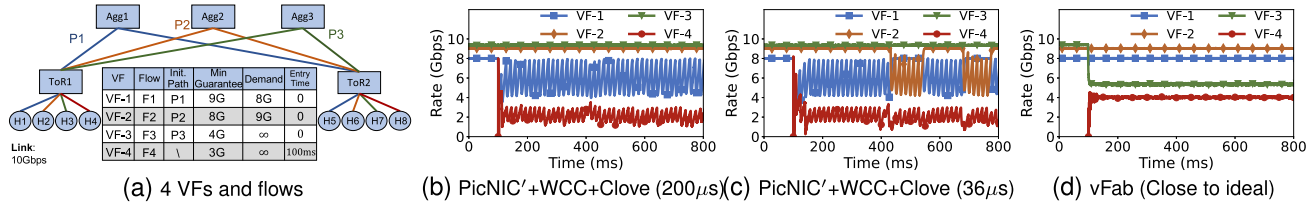


Fig. 4. Path migrations for utilization-oriented load balance may break bandwidth guarantees. (Case-3).

mitigates this, but it still converges slowly. Figure 3c shows that $F2$ finishes before it converges to its guarantee demand (7Gbps). This case demonstrates that the prior work cannot always guarantee bandwidth under work conservation, due to their impractical assumption (Figure 3b) or slow convergence (Figure 3c). In the following comparisons, we omit PicNIC′ and compare only with PicNIC′+WCC+Clove.

Case-3: Guarantee-agnostic load balance may break bandwidth guarantee. As shown in Figure 4a, $F1, F2, F3$ are initially allocated to different paths and achieve the desired bandwidth. At this point, the bandwidth subscription of Path $P1-3$ is 90%, 80% and 40%, respectively, while their utilization is 80%, 90% and 100%, respectively. Note that utilization is essentially different from subscription, due to insufficient traffic demand ($F1$) or work conservation ($F2$ and $F3$). At $100ms$, $F4$ enters into the network. Since path $P1$ has the lowest utilization, $F4$ is assigned to $P1$. But it causes the bandwidth dissatisfaction of $F1$. With the recommended flowlet gap (200μs) in Clove, $F1$ and $F4$ remain in $P1$, with non-guaranteed bandwidth. This result shows that the existing guarantee-agnostic load balance mechanism cannot perceive the breaking of bandwidth guarantee.

Furthermore, to force Clove functional in this case, we decrease the flowlet gap to 36μs ($1.5\times baseRTT$), so a slight queuing would trigger path migration. In this case, as shown in Figure 4(c), $F4$ never gets desired bandwidth. When $F4$ competes with $F1$ on $P1$, it finds out $P2$ has the lowest utilization and switches to $P2$. However, once $F4$ is assigned to $P2$, both $F2$ and $F4$ cannot obtain the desired bandwidth. Then $F4$ will try to move back to $P1$ again (since $P1$ has the lowest utilization now). $F4$ migrates paths under the false guidance, leading to network oscillations, and the cascading effect breaks bandwidth guarantees of other VFs (VF1 and VF2). It shows that a straightforward combination of existing solutions fails to provide bandwidth guarantee with work conservation, because path utilization cannot reflect the essential bandwidth contention, *i.e.*, total subscription of minimum bandwidth demand on a path.

C. Our Proposal: Informative Data Plane

In the traditional network architecture, typically, the edge (end-hosts) works independently with the network core (switches), causing that prior work commonly treats the core as a pipe with little direct feedback, either assuming an ideal core or leveraging heuristics to infer the network status.

Fundamentally, this issue can be solved if the network core can provide explicit information. With the help of commodity programmable switches and NICs, abundant in-network information, which is previously inaccessible, now can be conveniently calculated, stored, and transmitted. The informative data plane can allow the edge to make timely decisions on data transmissions without going through the time-consuming and inaccurate heuristics. For instance, if the core directly tells the edge that a link is experiencing gray failure, the latter can immediately (RTT-level time) reroute to a failure-free path via source routing, without requiring time-consuming route re-convergence in control plane.

This paper's core mission is to explore how to improve the VF dependability via collaboration between an informative core and an active edge. This direction looks promising – for instance, Figure 2, Figure 3d, and Figure 4d show that vFab's behaviors are close to ideal, in terms of both convergence speed and quality.

III. Design

This section presents the core design of vFab that builds a dependability framework on top of an informative data plane.

A. Design Goals and Assumptions

Service model. We abstract a VF using the *Hose Model* – a full-bisection fabric without internal capacity bottlenecks, where each VM in the VF should be able to send and receive with a minimum bandwidth pre-agreed with the tenant at any time. However, the traffic pattern within the VF, *e.g.*, many-to-one or one-to-many, determines the bandwidth for an individual VM-to-VM pair. As the construction of hose model is a well-studied area [18], [32], [40], we choose the idea of Guarantee Partitioning (GP) proposed by ElasticSwitch [15], which dynamically assigns the VM-pair bandwidth guarantees based on the hose model and real-time traffic patterns. Hence, we focus on the VM-pair bandwidth guarantees given by ElasticSwitch in this paper.

Design goals. vFab has the following design goals:

(i) Bandwidth guarantee with work conservation. A VF quickly provisions the minimum bandwidth for each VM-pair

that has sufficient traffic demand, while it also allows each VM-pair to swiftly go beyond the minimum bandwidth to fully utilize network resources if other VM-pairs do not have sufficient traffic demands;

(ii) Bounded tail latency. A VF bounds the end-to-end network latency for each VM-pair under bursty traffic;

(iii) Resilient reachability. A VF quickly restores reachability for each VM-pair, if failures occur in the underlay networks.

To make vFab as practical and deployable as possible, we also have some critical assumptions and non-assumptions:

Assumptions: *(i)* vFab operates on the DCN which is constructed by the commodity programmable switches, *e.g.*, Barefoot Tofino and Broadcom Trident-4; *(ii)* Similar to prior work [15], [34], [44], vFab assumes that VMs have been placed by other virtual cluster allocation algorithms, *e.g.*, Oktopus [32], so there are *theoretically* feasible solutions to satisfy the minimum bandwidth demands for all VFs even in the worst case.

Non-assumptions: *(i)* vFab does not assume a failure-free or congestion-free network core and DCN topology can either have over-subscription or not; *(ii)* vFab does not assume a perfect control plane or load balancing — the control plane can react slowly during failures and be agnostic to gray failures, and the load balancing can have hot spots due to reasons like hash polarization or hash collision; *(iii)* vFab does not assume any traffic patterns from the tenants. There can be unpredictable on-off traffic bursts, incasts, or long persistent flows with occasional microbursts. *(iv)* vFab does not assume a switch to have a large number of priority queues. Since queues are scarce resources and used for different purposes, vFab only needs a single queue.

B. System Overview

In this part, we first identify the necessary and sufficient telemetry data from the network core for VF dependability, and then illustrate the architecture and workflow of vFab.

Critical telemetry data. Through our deliberate analysis and evaluation, we summarize the critical telemetry data below.

(i) Link health. It provides explicit information about link failures, including gray failures, helping the edge to migrate to a failure-free path in time, *i.e.*, resilient reachability.

(ii) Link capacity. It explicitly guides path selection and rate control, since the capacity of switches may be different.

(iii) Queuing size. It reports the current queue status in switch ports. Edge can timely reduce sending rate to control queuing latency when observing a queue is building up.

(iv) TX rate. It reflects the output rate of the switch port. Combined with queuing size, edge can perceive the gap between actual link load and link capacity, allowing to accurately adjust rate to quickly converge to the target utilization, *i.e.*, work conservation.

(v) Total bandwidth subscription. It is the sum of the minimum bandwidth demands of all active VFs passing through the link, guiding edge to determine whether a path can satisfy its minimum bandwidth demand.

(vi) Total sending window. It represents the sum of the traffic admission windows of all active VFs passing through the link, which acts as a reference for weighted fair sharing.

Architecture. Figure 5 shows that vFab installs edge agents (vFab-E) and core agents (vFab-C) into the DCN

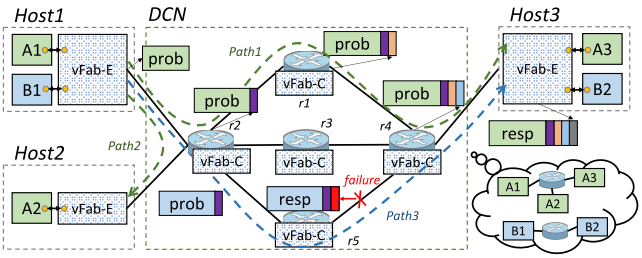


Fig. 5. The overview of vFab's system architecture.

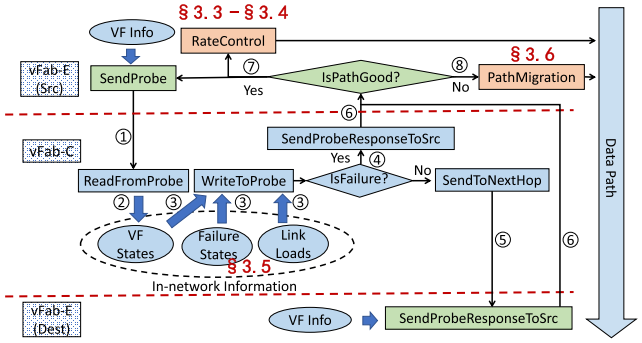


Fig. 6. The overall workflow of vFab.

edge and the DCN core, respectively. The two types of agents work collaboratively via periodic probe and the corresponding response, *e.g.*, *prob* from $Host_1$ and *resp* from $Host_3$. vFab ended up with two isolated VFs for two tenants, *i.e.*, A and B, A1 is a VM of A.

At edge, vFab-E aggregates one tenant's application flows from one VM to another into a moderate number of underlay network (directional) paths through tunneling or source routing. Source vFab-E inserts local VF information, *i.e.*, minimum bandwidth demand and sending window, into the probe. Along the forwarding path, the vFab-C put the aggregated VF information, *i.e.*, total bandwidth subscription and total sending window, and link information, *i.e.*, link capacity, queue size, TX rate, and link health (immediately responsing if any failure) into the probe via INT. Destination vFab-E returns all information piggybacked in the probe with a response, together with its local VF information.

Workflow. As shown in Figure 6, each vFab-E sends probes on the paths it uses (Step ①). After the probe reaches a vFab-C, the vFab-C first reads the piggybacked VF information and aggregates it with the internal VF information (Step ②), then inserts the updated result into the probe (Step ③). Depending on whether the outgoing link is experiencing failures or not, it will decide whether send back the probe response (Step ④) or forward to the next hop (Step ⑤). When the response sent by destination vFab-E or vFab-C gets back (Step ⑥), source vFab-E will decide whether to simply adjust the sending rate based on the information in the response (Step ⑦) or migrate path if the current path is not qualified anymore (Step ⑧).

C. Bandwidth Allocation

vFab combines the manners of distributed and dynamic bandwidth allocation and advanced congestion control to achieve strong bandwidth guarantee and work conservation.

Guaranteeing minimum bandwidth. We define $\phi_{a\to b}$ as the bandwidth token allocated by ElasticSwitch for VM-pair

$a \rightarrow b$ [1] and B_u as the minimum bandwidth a unit token can give to a VM-pair. Hence, the minimum bandwidth $a \rightarrow b$ can get is $B_{a\rightarrow b} = B_u \times \phi_{a\rightarrow b}$. The sender vFab-E needs to choose a path and control sending rate to satisfy $B_{a\rightarrow b}$ for VM-pair $a \rightarrow b$.

The strategy vFab takes is to share bandwidth proportionally to the bandwidth tokens of the VM-pairs. When multiple VM-pairs' underlay paths go through a link l with target bandwidth capacity C_l ,[2] they should share the link capacity proportionally to their bandwidth tokens. However, since a sender does not know about other senders coexisting on l, it needs information feedback from l to know its proportional share. Thus we have:

$$r^l_{a\rightarrow b} = \frac{\phi_{a\rightarrow b}}{\Phi_l} \times C_l \tag{1}$$

where Φ_l is the total token of all active VM-pairs on l, which is reported by the vFab-C with vFab-E's probe responds. Then, $r_{a\rightarrow b} = \min_{l\in p_{a\rightarrow b}}\{r^l_{a\rightarrow b}\}$, where $p_{a\rightarrow b}$ is the path VM-pair $a \rightarrow b$ uses. $r_{a\rightarrow b}$ gives a lower bound of bandwidth that VM-pair $a \rightarrow b$ can get from $p_{a\rightarrow b}$.

We choose the proportional sharing strategy because it provides an essential feature to make a quick judgment of path quality and avoid harmful impacts to innocent VFs. Specifically, if $C_l \geq \Phi_l \times B_u$, all VM pairs can be satisfied ($r_{a\rightarrow b} \geq B_u \times \phi_{a\rightarrow b}$). Otherwise, none of the VM pairs can achieve its minimum bandwidth. Therefore, if the sender a predicts that $C_l < (\Phi_l + \phi_{a\rightarrow b}) \times B_u$ after VM pair $a \rightarrow b$ joins in the link l, it will avoid using this path, because otherwise all existing VFs on l could be unsatisfied.

Work conservation. It is safe but not efficient if all senders' sending rates are up to $r_{a\rightarrow b}$, because some senders might not always have data to send, *i.e.*, insufficient traffic demands. For work conservation, senders should use $r_{a\rightarrow b}$ as a lower bound, and it can go beyond that. We define $R_{a\rightarrow b}$ as the upper bound of bandwidth that VM-pair $a \rightarrow b$ can have. Suppose link l's actual TX rate tx_l is lower than the target capacity C_l. In that case, all senders can scale up their sending rate. Thus the sending rate in Eqn (1) is replaced by:

$$R^l_{a\rightarrow b} = \min\{\frac{\phi_{a\rightarrow b}}{\Phi_l} \times R_l \times \frac{C_l}{tx_l}, C_l\} \tag{2}$$

where $R_l = \sum_{\forall a\rightarrow b\in P_l} R^l_{a\rightarrow b}$ is also reported by vFab-C in probe responses. In Eqn (2), $\frac{C_l}{tx_l}$ can precisely measure the gap between the actual and the target link utilizations. By reporting the centralized view (Φ_l and tx_l) to the edge, a core link guides the senders to scale up (*e.g.*, switching from bandwidth guarantee to work conservation) or down (*e.g.*, switching from work conservation to bandwidth guarantee) to approach the target utilization. Similarly, we have $R_{a\rightarrow b} = \min_{l\in p_{a\rightarrow b}}\{R^l_{a\rightarrow b}\}$.

D. Traffic Admission

Avoiding queuing in the core : Eqn (2) only considers bandwidth convergence, but short-term traffic bursts can happen and cause latency spikes in transient. To control the queuing latency in the core, we advocate the window-based flow control, which are widely used in TCP and RDMA [45], [47]. Then, we modify Eqn (2) to the following:

$$w^l_{a\rightarrow b} = \min\{\frac{\phi_{a\rightarrow b}}{\Phi_l} \times W_l \times \frac{C_l \times T_{a\rightarrow b}}{tx_l \times T_{a\rightarrow b} + q_l}, C_l \times T_{a\rightarrow b}\} \tag{3}$$

where $T_{a\rightarrow b}$ is the $baseRTT$ between a and b without queuing; q_l is the real time queue size of l; $W_l = \sum_{\forall a\rightarrow b\in p_l} w^l_{a\rightarrow b}$ is the total sending window of all active VM-pairs traversing link l. Then, $w_{a\rightarrow b} = \min_{l\in p_{a\rightarrow b}}\{w^l_{a\rightarrow b}\}$ is the sending window size of VM-pair $a \rightarrow b$. Eqn (3) controls the total inflight traffic and reduces the sending windows when the queue of link l is building up. Essentially, vFab converges to weighted fairness rapidly while maintaining close-to-zero queuing latency with the information of Φ_l and W_l from the core. The theoretical analysis on vFab's fairness and utilization convergence is in Appendix A.

Bounding the worst-case latency : While $w_{a\rightarrow b}$ allows immediate use of network capacity if the window permits, it cannot handle transient congestion during synchronized bursts (*e.g.* incast). Especially, when the link keeps being underutilized, it is quite possible that $w_{a\rightarrow b} = C_l \times T_{a\rightarrow b}$ which means any VM pair with a single token can use the full capacity. Then, if multiple VM pairs have traffic demands simultaneously, the total inflight traffic and worst-case latency are still unbounded.

Our strategy is two-stage traffic admission, allowing tenants to ramp up to its minimum bandwidth demand quickly and a little slower to converge to work conservation. This is because tenants pay for its bandwidth demand, but the extra capacity is a "bonus", and doing this can provide a strict bound to the inflight traffic and so as the end-to-end latency.

Scenario-1 : For a new VM-pair just joining a path, it uses $w'_{a\rightarrow b} = \phi_{a\rightarrow b} \times B_u \times T_{a\rightarrow b}$ as bootstrap sending window. It then performs additive increasing by $w'_{a\rightarrow b} \leftarrow w'_{a\rightarrow b} + \frac{\phi_{a\rightarrow b}}{\Phi_l} \times C_l \times T_{a\rightarrow b}$ per RTT until $w'_{a\rightarrow b}$ is larger than the $w_{a\rightarrow b}$ from Eqn (3) and then start to use $w_{a\rightarrow b}$.

Scenario-2 : For an existing VM-pair on a path but with actual traffic demand lower than $r_{a\rightarrow b}$, it first uses $w'_{a\rightarrow b} = r_{a\rightarrow b} \times T_{a\rightarrow b}$ and then follows the same increasing procedure as in *Scenario-1*. Because on a qualified path, $C_l \geq B_u \times \Phi_l$, *Scenario-2* creates the same or more load (utilization) than *Scenario-1*.

For a link l, in the worst case, all VM pairs want to send traffic simultaneously (in *Scenario-2*) and increase one BDP per RTT. Our theoretical analysis (in Appendix A) suggests senders take *2 RTTs* to learn the load created by the incast from the core and start to reduce its sending rate, the maximum inflight bytes is bounded by $3 \times BDP$:

$$\sum_{\forall a\rightarrow b\in P_l} r^l_{a\rightarrow b} T_{a\rightarrow b} + \sum_{\forall a\rightarrow b\in P_l} \frac{\phi_{a\rightarrow b}}{\Phi_l} C_l T_{a\rightarrow b} + tx_l T_{max} < 3C_l T_{max} \tag{4}$$

where T_{max} is the maximum $baseRTT$ (diameter) of the DCN.

Summarizing bandwidth demands in vFab-C : While tx_l, q_l, and C_l are straightforward to obtain by programmable switches [45], Φ_l and W_l are essential summaries of bandwidth demands submitted from the edge and computed in the core. Maintaining the runtime Φ_l and W_l on the switch is nontrivial. It requires the switch to know when a VM-pair is active or inactive, which is almost impossible to

[1] § III-G discusses how to distribute VM's tokens among VM-pairs.

[2] $C_l = \eta C'_l$. C'_l is l's physical bandwidth capacity and we pick $\eta = 0.95$.

be directly recognized by switch. Rather than entangle the switch processing logic, we maintain necessary status on the edge. First, the probe of VM-pair $a \rightarrow b$ takes its current $\phi_{a\rightarrow b}$ and $w^l_{a\rightarrow b}$, so that each switch reads them when the probe bypasses the switch. A switch maintains two registers for Φ_l and W_l. It uses a Bloom filter to check whether a VM-pair's $\phi_{a\rightarrow b}$ and $w^l_{a\rightarrow b}$ have been seen. If not, it will add $\phi_{a\rightarrow b}$ and $w^l_{a\rightarrow b}$ to the two registers and record the VM-pair in the Bloom filter. A VM-pair will explicitly tells all switches via a finish probe when it becomes inactive: either it is idle for a while or leaves the current path. Thus, the switches along the path can adjust Φ_l and W_l in the Bloom filter. The VM-pair will not stop sending the finish probe until it gets the acknowledgments from all switches in the probe response. vFab-C also handles silent quits (§IV-B).

The occasional false positive of Bloom filter has limited impacts. If a false positive happens, a VM-pair will be omitted such that Φ_l and W_l will be smaller than the truth. While it will increase $r_{a\rightarrow b}$ a little bit, it does not influence the proportional sharing and work conservation. A larger $r_{a\rightarrow b}$ means the VM-pair has an enormous burst at first, and some VM-pairs will choose a path that indeed has no resource to serve the bandwidth demand. But the small headroom of the link capacity (5% in our implementation) and the path migration due to bandwidth dissatisfaction will digest these cases. Also, vFab-C is open to leverage other advanced streaming algorithms, such as timing Bloom filter [48].

E. Failure Detection

Device failures are common in DCNs, including fail-stop failures and gray failures [9], [10]. It takes seconds to minutes for the control plane to bypass the faulted devices [12], [13], and even the data plane's solutions will take milliseconds [23], [24], [25]. During the recovery period, the bandwidth demands of the affected VM pairs cannot be guaranteed, and packet loss can lead to a spike in latency. Worse, blindly rerouting within the network will break the stable state of other paths and cause traffic oscillations. For example, after rerouting, the capacity of the new path cannot meet the total bandwidth demands, leading to further path migration (§III-F). In order to quickly recover the reachability and smartly select a good enough path, we follow the idea that *vFab-C is responsible for quickly detecting failures and notifying vFab-E, which is responsible for path migration*.

Although the programmable data plane provides the capabilities to access the packet losses [42], achieving fast failure detection in the data plane is extremely challenging. Firstly, the causes of failures are complex and varied, and there are even unknown types of failures. Thus, deploying loss counters case by case to detect failures is inefficient and incomplete. Secondly, if we detect packet losses independent of their causes, *i.e.*, count the difference between the total number of forwarded packets at both ends of a link, like LossRadar [49], non-fault packet loss (*e.g.* congestive loss, ACL discard) will give us the wrong indication. So, the ability of vFab-C to fully cover the fault packet loss is the key to achieving resilient reachability.

Placing meters : vFab-C leverages the cooperation of two connected switches to achieve fast and accurate failure detection for both fail-stop and gray failures. Figure 7a shows how vFab-C places the meters to cover a link between two switches. vFab-C deploys two arrays of meters: 1) Upstream meters (UM) are placed at the beginning of the egress pipeline in the upstream switch; 2) Downstream meters (DM) are placed at the end of the ingress pipeline in the downstream switch. Each meter array contains one meter that counts the number of packets that pass through it. Obviously, the difference between UM and DM is the number of packets lost on this link, because the *Shared Buffer* is not covered, and therefore packets lost due to congestion are not counted.

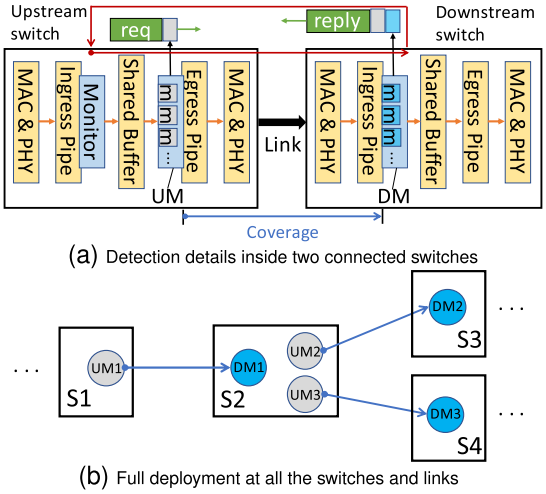


Fig. 7. vFab-C's failure detection. Placing meters to cover the entire network excluding non-fault losses.

In addition, each meter array has some meters to count the number of non-fault lost packets in pipeline (*e.g.*, TTL expiration or ACL rules), because they should be ignored when determining whether a failure has occurred. Finally, vFab-C covers the full range of packet loss due to failure that occurred within the segment between UM and DM, ranging from faulty physical components (*e.g.*, dirty cable, link flapping) to control plane oscillation (*e.g.*, L3 lookup miss due to BGP oscillation).

We use an example to show how to compute the packet loss rate. There are 100 packets passed through the UM and went out (*i.e.*, the UM will be [100, 0]), and 8 of them were lost on the physical link and 2 of them were lost because of the ACL rules in the ingress pipeline of the downstream switch. The DM will be [90, 2]. By comparing the UM and DM, the packet loss rate of the link should be 8% rather than 10%.

Figure 7b shows the network-wide deployment of the failure detection mechanism. Every link is covered by one pair of UM and DM. Note that although we do not draw in Figure 7b, there is also traffic from S2 to S1, so actually we also need an UM at S2's output port connected to S1, and a DM at S1's input port connected to S2. In fact, we need two pairs of UM and DM for each bidirectional link.

Comparing the UM and DM : As shown in Figure 7a, *Monitor* module in the upstream switch will send out an echo-request to the output link. The echo-request piggybacks both UM and DM via telemetry [50], [51]. Then, it will be bounced back as an echo-reply at the downstream switch. Upstream *Monitor* processes the echo-reply, and computes the packet loss rate. Missing echo-reply can be used to detect the output link's malfunction (*i.e.*, 100% loss rate). Finally, *Monitor* marks the output link as a failure if the loss rate is beyond a predefined threshold (*e.g.* 5%) for a sufficiently long time. When a probe sent by vFab-E will pass over the faulted output link, we immediately bounce it back with a fault

flag within the data plane. § IV-B describes the workflow in detail.

Batch synchronization : To make a correct comparison, the UM and DM need to count the same batch of packets. It looks like we need a precise time-synchronization protocol, but in fact, we just need to reset the meters after the echo-request packet reads its data. Because the two directly connected ports do not have packet disorder, the sequence of data packets and echo-request packets is same at the UM and DM, and the echo-request packet can provide the build-in batch synchronization. Indeed, the loss of echo-request packets will break the batch synchronization, but this can be avoided. We configure Active Queue Management [52] to avoid dropping them when buffer is very congested. If they encounter a fault loss, *Monitor* will not receive a reply and consider these is a failure.

Recovery detection : After manual or automatic repairs, we still use meters to test whether the port is healthy. To this end, *Monitor* periodically sends a batch of test packets that will pass through the UM and DM. An echo-request packet is also used to collect the meters. Finally, *Monitor* marks the output link as healthy if the packet loss rate is below a predefined threshold for a long time. vFab-E can learn this through path probe.

F. Path Migration

While vFab-C is able to report the quality of path, vFab-E needs to avoid the side-effect caused by path migration.

Triggers of path migration : In vFab there are three major reasons for path migrations: (i) Avoiding failures to restore reachability; (ii) Satisfying minimum performance requirements when the current path becomes incapable; and (iii) Obtaining more resources from idle paths for network-wide work conservation. We have different strategies for them to enhance the overall stability of the network, due to their descending emergency. (i) has the highest priority, and it is executed right after failures are discovered; (ii) should be done quickly with cautious – in order to avoid unnecessary disturbs to the network, a VM-pair should monitor for a sufficiently long time (5 RTTs in our implementation) to ensure that the current path is consistently violating minimum bandwidth guarantee; and (iii) should be performed even less frequently – a VM-pair should observe a persistently better path for a long duration (30 seconds in our implementation) before migrating.

Path selection : When a VM-pair initially joins in or begins a path migration, vFab-E sends multiple probes to different paths in parallel to bootstrap or to start a migration. It marks all paths which can serve with minimum bandwidth guarantees, *i.e.*, $C_l \geq (\Phi_l + \phi_{a\to b}) \times B_u$ on all links, as "qualified". Among all qualified paths, it selects one randomly with a preference to the path with minimum bandwidth subscription. For the migration trigger (iii), only the "qualified" path with the largest $R_{a\to b}$, is considered.

Avoiding oscillations : The synchronization may cause oscillation in the edge: a congested path is given up by all the VM-pairs on it, which becomes idle later, while all the VM-pairs have high probability to select the idle path together, making it congested soon. If the minimum bandwidth guarantee is violated, vFab-E only allows one path migration in a randomly picked freeze window within $[1, N]$ RTTs on each host. Hence, a vFab-E agent waits for at least one RTT to see the load change after the last migration, which is essential to the convergence as pointed by [53] and our evaluation.

Avoiding reordering : Though not all tenants will require to prevent packet reordering during path migrations, vFab still offers an option to do that. If this option is enabled, vFab-E will only probe without sending data in the first RTT on the new path, allowing the packets on the old path to be cleared.

G. Discussions

Explicit bandwidth allocation : vFab obtains network state and VF demands from the core and calculates bandwidth allocation at the edge. An alternative division of labor is that the core explicitly maintains a *base rate*, and the edge allocates bandwidth by multiplying the *base rate* by its bandwidth token, *e.g.*, weighted RCP [54]. While it is helpful for weighted bandwidth sharing, it fails to find a proper path for a VF. Also, it is challenging for today's commodity programmable switches to support such complicated arithmetic operations.

Bandwidth token assignment : vFab is open to any bandwidth tokens assignment algorithms that dynamically assigns VM's tokens to VM-pairs. For concreteness, vFab adopts the idea of GP in ElasticSwitch [15]. vFab partitions a VM's token among its remote peers, and the unused tokens contributed by bounded VM-pairs, *i.e.*, traffic demand is insufficient, are reassigned to unbounded ones. We use a per-VM-pair rate meter to estimate demand. While the algorithm is not novel, Appendix B describes it in detail for completeness.

The number of underlay paths : The traditional path management in DCN spreads the flows into numerous underlay paths, while vFab only assigns a small number of underlay paths (even a single path in the majority of this paper) for a VM-pair. Obviously, vFab's way has smoother traffic and is more manageable. Also, vFab is responsive to capacity issues, because its underlay paths are dynamic to protect performance. In high bisection DCNs, vFab can even use a single dynamic path for each VM-pair, while it indeed needs more paths in oversubscribed DCNs to utilize more end-to-end bandwidth. Appendix C shows how vFab scales to multiple underlay paths.

IV. Implementation

We fully implement vFab with SmartNICs and programmable switches. We have two versions for the active edge: one is on ARM-based SoC SmartNICs (Broadcom Stingray PS225), and the other is on FPGA-based SmartNICs (Xilinx Alevo U200). The SoC version can fully support any transport stack and application software transparently, while the FPGA version is used to evaluate vFab's complexity, efficiency, and performance running in hardware.[3] The SoC smart NIC consists of eight 3.0 GHz embedded processors, 16 GB DRAM, and PCIe Gen3 $\times$ 8 interface. It runs CentOS 7 OS and uses DPDK 17.11.4 at bare-metal mode for inline packet processing for a 10G port. The FPGA smart NIC provides line-rate packet processing for a 100G port. It has a 64 GB onboard DRAM and a PCIe Gen3 $\times$ 16 interface. The implementation has 8000+ lines of C++ code in the SoC version and 18000+ lines of Verilog code in the FPGA version.

In the informative core, we implement vFab-C in a Tofino programmable switch with 32×100G ports. It has a CPU of four 2.2 GHz cores, 16 GB DRAM, and a PCIe Gen2 $\times$ 4 bus.

[3]The FPGA version has not transparently supported socket-based applications yet due to engineering reasons.

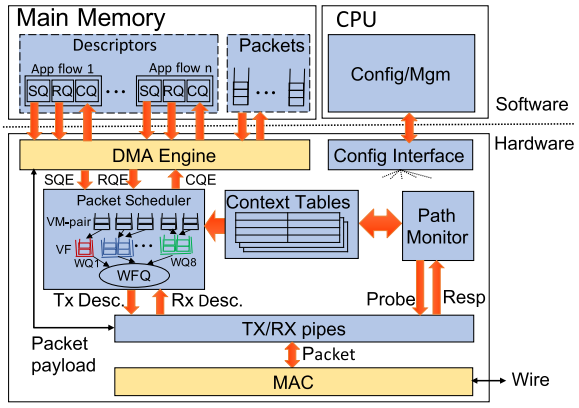


Fig. 8. vFab-E on FPGA-based SmartNIC.

The implementation uses about 3400 lines of P4 code and 3600 lines of python configurations in the control plane.

A. vFab-E at SmartNICs

The following descriptions focus on the FPGA-based implementation. There is a slight difference that vFab-E in FPGA supports Verbs interface [55] with DMA, while we use DPDK in SoC to transparently support socket-based transports.

Figure 8 shows the vFab-E's overview to leverage several modules to perform traffic admission and path selection. *Packet Scheduler* maintains queues for each VM-pair to limit the inflight traffic and avoid head-of-line(HoL) blocking. VM-pair queues belonging to the same VF are grouped together as a weighted VF queue. All VF queues connect to a WFQ engine to enforce the weighted fair-sharing across tenants at the sender side. Thus, *Packet Scheduler* runs a hierarchical traffic admission by both VM-pairs' sending window and tenant-level WFQ. *Context Tables* maintain the states for active VM-pairs. Most of these states are used to construct the packet headers, while some are used to record the path quality and status, such as reachability, bandwidth token, sending window, and latency bound. *Path Monitor* maintains the reachable paths discovered by the existing route discovery component, *e.g.* source routing (SR) controller, and continuously monitors the path quality and performs path migration.

vFab-E workflow : For each new VM-pair, the software first configures the VM-pair states into *Context Tables* and obtains an initial path from *Path Monitor* towards the destination. Packet path is enforced by SR, which includes output port ID of each hop [31]. The software then follows the standard Verbs APIs to exchange packets and descriptors with vFab-E via DMA. *Packet Scheduler* first uses WFQ engine to schedule a next VF to send. It then scans through all VM-pairs in this VF in a round-robin manner to schedule a VM-pair permitted by the VM-pair's sending window. Similarly, it sends one packet from the application flows belonging to the scheduled VM-pair in a round-robin manner. Next, the corresponding TX descriptor is passed to TX pipe to fetch the actual packet via DMA to emit. On receiving a packet, RX pipe directly delivers the packet to the main memory via DMA without software involvement. *Path Monitor* will send probe in two cases, one is to periodically monitor the paths in use, the other is to detect other paths for path migration. Consecutive network dependability violations or drops of probes will trigger path migration. Since the end-to-end latency is bounded

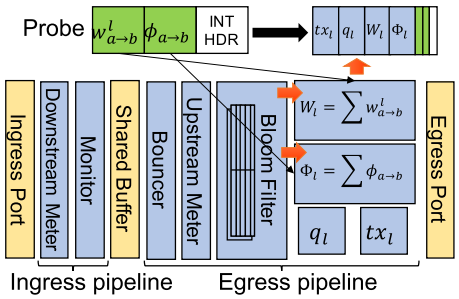


Fig. 9. vFab-C on P4 programmable switch.

(4 $baseRTT$s), we detect probe loss by timeout beyond 8 $baseRTT$s.

Scalable probing scheme : An intuitive probing scheme is a probing loop: sending next probe right after receiving the previous response. But it causes the probing overhead to increase linearly as the number of VM-pairs increases. Instead, vFab sends probes only when a VM-pair has traffic demand. Specifically, after receiving the previous response, the VM-pair sends the next probe only after transmitting a predefined amount (L_w) of traffic (*e.g.*, 5× MTU). In the worst-case where many VM-pairs send traffic simultaneously, the probe overhead is at most $\frac{L_p}{L_p+L_w}$ of the bandwidth, where L_p ($L_p \ll L_w$) is the size of the probe packet. In practice, this scalable scheme still retains a good track of information in the core.

Scaling to a large number of VFs : Building a hierarchical WFQ engine for massive VFs is very resource consuming: 1) each weighted queue requires an independent block RAM unit; 2) scheduling multiple queues requires the implementation of the N-channel multiplexer, whose cost grows super-linearly. To this end, we constrain the WFQ engine to use only 8 weighted queues (colored queues in Figure 8) with distinct levels of weights for VFs to select. Different VFs in the same weighted queue are scheduled in round-robin. This implementation provides the same weighted scheduling results. Using constraint weights slightly limits the performance differentiability but greatly improves the scalability and cost-efficiency. We believe it is a good trade-off in practice.

Resource usage : The prototype of vFab-E on FPGA supports 8K VM-pairs and 1K tenants, using only less than 10% of the resources (*e.g.* Registers, LUTs, Block RAM, *etc.*).

B. vFab-C at Programmable Switches

Figure 9 shows the system structure of the informative core implemented on a P4 programmable switch.

Telemetry : As described in § III-B, vFab-C reads the VF's demand ($w^l_{a\to b}$ and $\phi_{a\to b}$) from the probes and writes the link status (tx_l, q_l, and C_l) and demand summaries (W_l and Φ_l) back to the probes. In a DCN with a maximum of 5 hops, the total size of the telemetry data is less than 100 bytes, which has low overhead. Appendix D shows a specific format of probe/response packet.

Information summary : To recognize active VM-pairs for computing Φ_l and W_l, vFab-C adopts a Bloom filter with four memory banks running in parallel. With a 4-way hashing Bloom filter of 15 KB, vFab-C supports a moderate of 20K distinct VM-pairs with less than 5% false positives, while those false positives have limited impacts as presented above.

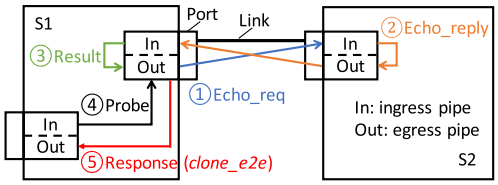


Fig. 10. Workflow of failure detection.

TABLE II
RESOURCE CONSUMPTION OF vFab-C PROTOTYPE ON INTEL BAREFOOT TOFINO. (vFab-C-: vFab-C WITHOUT THE FAILURE DETECTION)

Resource Type	vFab-C-	vFab-C
Match Crossbar	8.64%	8.64%
SRAM	18.75%	19.88%
TCAM	6.25%	6.25%
VLIW Actions	18.23%	18.23%
Hash Bits	17.07%	17.63%
Stateful ALUs	47.92%	48.38%
Packet Header Vector	20.05%	20.72%

Handling silently inactive VM-pairs : vFab requires each VM-pair to explicitly notify its activity to switches, while a VM-pair may become inactive silently due to unexpected behaviors. This causes Φ_l and W_l to be larger than expected. To this end, vFab-C periodically (10 seconds in our implementation) cleans inactive items, *i.e.*, no probe is received in the last period, in the Bloom filter and decreases Φ_l and W_l.

Failure detection : UM (and DM) consists of three meters, so UM's size is 24 bytes. The first meter counts the data packets traversing it. The second meter counts the lost packets due to TTL expiration. The third meter counts the lost packets due to ACL rules. *Monitor* is responsible for comparing UM and DM, and *Bouncer* in the egress pipeline is responsible for bouncing the probe if it is a failure.

Figure 10 shows the workflow of the failure detection mechanism. ① In every 100 μs, *Monitor* in S1's ingress pipeline sends out an echo-request packet to the egress pipeline in the same port, using the *packet generation engine* in Tofino ASICs [56], [57]. The format of the echo-request packet is similar to the LLDP protocol, and we add the INT header to store the value of meters. ② *Monitor* in S2's ingress pipeline detects the echo-request packet, and modifies it as echo-reply, and sends it to S1. ③ *Monitor* in S1 handles the echo-reply packet. If the loss rate is bigger than 5%, it shares the failure information to the egress pipeline. ④ The probe from a port is routed to the output port. ⑤ *Bouncer* bounces back the probe as a response if it is a failure. To do this, we use the clone egress to egress or *clone_e2e* primitive provided by programmable switching ASICs [58], [59]. Note that the response packets use IP-based routing, instead of SR.

Resource usage : The resource consumption of vFab-C and the failure detection mechanism is shown in Table II. The failure detection mechanism requires only 64 bytes of SRAM per port to store meters, with minimal consumption of other resources. Besides, the generation rate of the echo-request is 10Kpps, the size of the packet is $<$ 200 bytes, thus the bandwidth overhead is only 2Mbps per port. In summary, the processing overhead in vFab-C is very small and has no impact on the line-rate processing of other data packets.

V. EVALUATION

We use testbed experiments with our prototypes and large-scale NS3 simulations [60] to evaluate vFab performance.

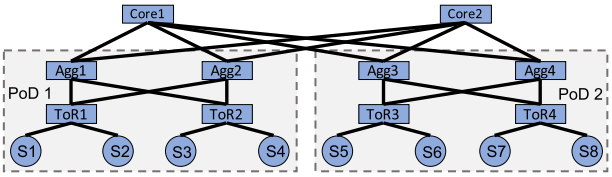


Fig. 11. Testbed topology.

A. Evaluation Setup

Environment : As shown in Figure 11, our testbed is a 3-tier topology with two Pods, which contains 8 servers (S1-S8) and 10 programmable switches. Each server has a 96-core Intel Xeon 2.50GHz CPU, 791 GB memory, and two NICs – one SoC smart NIC with a 10G port and one FPGA smart NIC with a 100G port. For all experiments, we set the target bandwidth utilization as 95%. The maximum network $baseRTT$ is 24 μs. The default token update period is set as 32 μs. The network topology in NS3 simulations contains 512 servers connected in a FatTree [61] with 100Gbps links. We use 16 and 32 Core switches to set an oversubscription ratio of 1:2 and 1:1, respectively.

Alternatives : We compare vFab to two kinds of combination of existing solutions, *i.e.*, PicNIC′+WCC+Clove and ElasticSwitch+Clove, since the combinations can be used for end-to-end network performance of multi-tenant DCNs.

B. Microbenchmarks

We first run our SoC prototype in the testbed and use micro-benchmarks to compare vFab with alternatives.

Bandwidth guarantee with work conservation : This experiment shows how vFab and the alternatives handle intensive minimum bandwidth demands. For this, a permutation traffic pattern with different bandwidth demands is generated. There are three classes of VFs: minimum bandwidth demand of 1Gbps, 2Gbps, and 5Gbps, respectively. Each VF has only one VM-pair whose source is in PoD-1 and destination is in PoD-2, so all traffic traverses Core switches. Each host has one VF per class, ensuring that the traffic is not bounded at the host ($1 + 2 + 5 = 8$Gpbs $<$ 10Gbps). We randomly insert a VF every 20 ms to evaluate the convergence speed and bandwidth guarantee properties.

Figure 12a shows that vFab achieves fast convergence when a new VF joins in. It persistently guarantees the minimum bandwidth demands with work conservation for all VFs via distributed traffic admission and path migration. In contrast, PicNIC′+WCC+Clove (Figure 12b) converges slowly when a new VF joins in, and suffers from rate fluctuation when all VFs coexist. Hence, it only guarantees 3 VFs' minimum bandwidth. ElasticSwitch+Clove (Figure 12c) guarantees bandwidth for more VFs, but causes serious queuing in the network (Figure 12e), because ElasticSwitch uses the minimum bandwidth demand as a lower bound of sending rate, even if the network is congested.

Figure 12d shows the bandwidth dissatisfaction ratio, which is the amount of bandwidth violation over the total bandwidth demands. Both PicNIC′+WCC+Clove and ElasticSwitch+Clove fail to meet bandwidth guarantee for some VFs (more than 40% and 10%, respectively), because without explicit in-network information, they are difficult to find proper paths for VFs. On the contrary, vFab efficiently adjusts VFs to proper paths and rapidly converges to steady

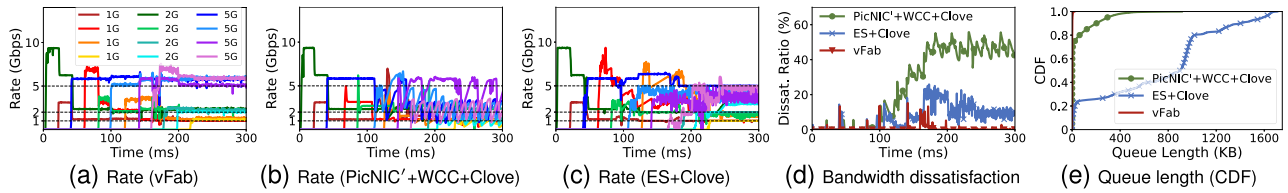


Fig. 12. Bandwidth evolution under high load. Bandwidth dissatisfaction indicates total minimal bandwidth violation.

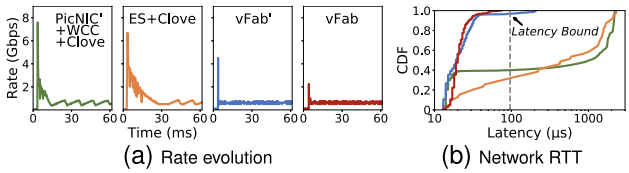


Fig. 13. Incast performance (bounded latency). Latency bound is calculated via $3BDP/C_l + baseRTT$.

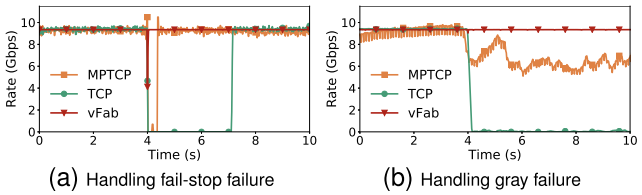


Fig. 14. Reachability and throughput under failures.

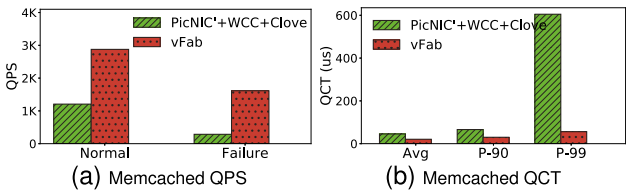


Fig. 15. Performance of Memcached.

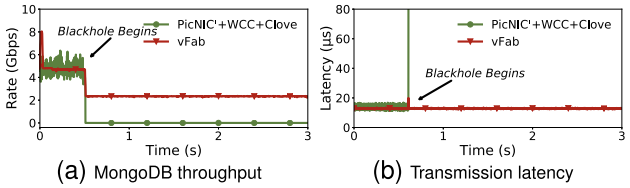


Fig. 16. Application performance under network failure.

bandwidth sharing. As a result, the dissatisfaction ratio is mostly close to zero even when VFs continuously join. Figure 12e shows that vFab always keeps queuing size low even with the continuous injection of new VFs, due to its excellent performance in terms of convergence speed.

Bounded latency : We extend the 14-to-1 incast experiment in Case-1 with more baselines. Figure 13a shows the rate evolution. vFab′ is the one without the optimization of *bounding the worst-case latency*. Both vFab and vFab′ quickly react to incast, and all VFs efficiently converge to the steady rate, while both PicNIC′+WCC+Clove and ElasticSwitch+Clove converge slowly with rate fluctuation. Figure 13b shows network RTT measured in the experiment. PicNIC′+WCC+Clove and ElasticSwitch+Clove have a 99th percentile RTT of $2.2ms$ and $2.3ms$, respectively, due to their slow reaction, while vFab′ decreases it by $11\times$ with feedback from the informative core. Still, these schemes cannot bound tail latency. With the optimization, vFab can restrain queuing and control tail latency under the expected bound.

Resilient reachability : Since most previous work does not take network failure into consideration, we select MPTCP as the comparison, which is a practically deployed transport layer protocol supporting failure recovery. Figure 14a and 14b show the behaviors of TCP, MPTCP, and vFab during fail-stop and gray failures, respectively. A failure happens at the Agg-to-Core link when a VM-pair of a VF with unlimited traffic traversing through it. During the fail-stop failure, which lasts for 3 seconds, TCP completely stops, and MPTCP shifts traffic to other paths after multiple RTOs. With the gray failure of dropping 5% packets, TCP drops to about 50Mbps and MPTCP's throughput decreases since it cannot confidently differentiate non-congestive loss to exclude the path. Instead, vFab relies on the core's explicit failure information to migrate to a failure-free path swiftly. This case also shows that applications can enjoy the resilient reachability on top of vFab without porting to multi-path transports by themselves.

C. Application-Level Performance

We evaluate application-level performance with transparent support of vFab, even under dynamic network situations. We set up two tenants: one tenant creates a VF to run Memcached, a latency-sensitive application, while the other creates another VF to deploy MongoDB, which is bandwidth-hungry. Memcached places 24 VMs as servers evenly over S7-S8, and 12 VMs as clients evenly over S1-S4. MongoDB places 24 VMs as servers evenly over S5-S8, and 24 VMs as clients evenly into S1-S4. Each Memcached client periodically fetches data from random servers, where the data size follows an empirical distribution of key-value workload [62] with a mean size of 2KB. Each MongoDB client continuously fetches 500KB data from a random DB server. Hence, the tenants compete for bandwidth in both edge and network core. Similar to [17], we focus on Memcached's performance due to its vulnerability to bandwidth contention.

Performance under normal network. vFab transparently supports the tenants with dependable VF on bandwidth and latency. Figure 15 shows the QPS (Query per Second) and QCT (Query Completion Time) of Memcached requests. vFab achieves $2.39\times$ higher QPS and $10.8\times$ lower tail QCT than PicNIC′+WCC+Clove, because the latter reacts to fabric congestion slowly and fails to find proper paths for different VFs.

Performance under network failure. We reproduce a switch routing black-hole incident from a production cloud (see §II-A). We inject a 3-second routing black-hole on Core1 switch. Figure 15a shows vFab's QPS is cut to half during failures, as Core layer lost 50% capacity, by migrating victim VFs away from failed paths. However, QPS of PicNIC′+WCC+Clove drops to about a quarter, and some clients even encounter application crashes due to the temporary loss of reachability. Figure 16a shows the timeline of MongoDB traffic from one server. When the failure occurs, vFab retains a predictable bandwidth from transparent path migrations without triggering perceivable impact on the upper-level

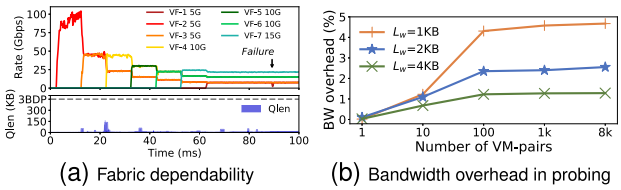


Fig. 17. Fabric dependability and overhead in 100GE.

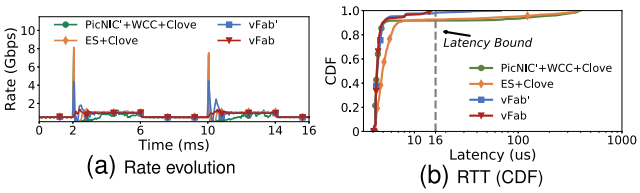


Fig. 18. Performance with 90-to-1 dynamic workload.

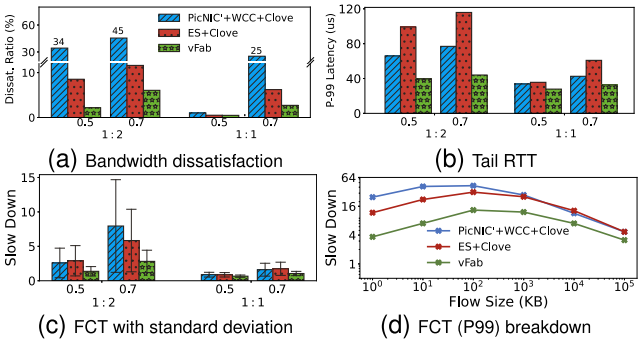


Fig. 19. Performance under real workload with 1:2, 1:1 over-subscription network and loads of 0.5 and 0.7.

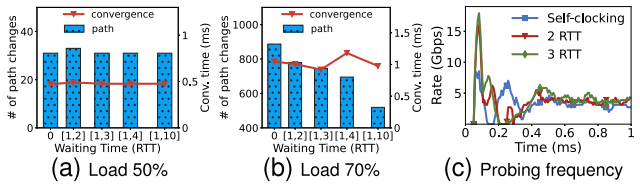


Fig. 20. Convergence and stability.

transport and application. However, PicNIC′+WCC+Clove cannot route around the black-hole, making MongoDB's throughput drop to zero. We further provide the end-to-end transmission latency during network failure. As shown in Figure 16b, at the moment of failure, the transmission latency encounters a slight and temporary increase, because vFab-E quickly recognizes the network failure and migrates impacted traffic to other paths.

D. Hardware Performance and Scalability

We use the FPGA-based prototype to show that vFab is capable of supporting line-rate of 100GE with small overhead.

Support 100GE network at line-rate : Figure 17a shows that vFab scales naturally to support 100GE and ensures predictability upon traffic dynamics and failures. 7 VFs towards S8 but with different minimum bandwidth guarantees continuously enter the network every 10 ms. vFab instantly converges to guarantee minimum bandwidth for all VFs while providing bounded latency during the rate transition. Surprisingly, when Core1 switch fails at 90th ms, the victim VFs cannot obtain the guaranteed bandwidth temporarily, and vFab swiftly migrates victim VFs to other paths. Despite the bursts and failures, vFab continuously maintains a close-to-zero queue.

Bounded bandwidth overhead of probing : We use one VF to generate persistent traffic to saturate the outgoing link and measure the bandwidth overhead using probes. As shown in Figure 17b, with the increasing number of VM-pairs, bandwidth overhead gradually goes steady (as suggested in § IV-A). By setting $L_w = 4$KB, the upper bound of overhead is 1.28%. Therefore, vFab can smoothly scale up to support more VM-pairs.

E. Convergence in Large Scales

We perform simulations to show vFab's convergence at scale.

Convergence under highly dynamic workload: We set a 90-to-1 scenario to show the performance of vFab under the extreme dynamic workload. All VFs have 1Gbps minimum bandwidth demand and periodically switch between fixed 500Mbps sending demands (underload) and unlimited sending demands every 4 ms. Figure 18 shows that PicNIC′+WCC+Clove suffers from great overshoot, leading to significant under-utilization. While ElasticSwitch+Clove quickly recovers to the minimum bandwidth demand, this aggressive behavior worsens latency. In contrast, both vFab and vFab′ can quickly converge to steady rate allocation. With latency optimization, the maximum RTT of vFab is bounded within 16μs, 27× lower than PicNIC′+WCC+Clove.

Performance under real workload: We generate tenant VFs with random minimum bandwidth demands. The number of VMs per tenant and the number of destinations each VM communicates at runtime are synthesized from empirical production data centers [63]. The distribution of flow size is consistent with empirical workload [20], and the average link loads are set to 50% and 70%, respectively. In each workload, over 40% of flows are expected to be completed within 100μs, if the minimum bandwidth is guaranteed. We make sure the minimum bandwidth of all VFs can be theoretically satisfied under these loads with Silo [34]. Figure 19a shows that the bandwidth dissatisfaction of vFab is much lower than the baselines, especially under heavy workloads. Though ElasticSwitch+Clove provides lower bandwidth dissatisfaction than PicNIC′+WCC+Clove by setting sending rate not lower than the minimum bandwidth demand, it causes higher tail latency (Figure 19b). In contrast, vFab efficiently guarantees the minimum bandwidth while controlling the network congestion, because it can adjust sending rate and select path accurately and rapidly. Hence, vFab achieves much better FCT slowdown,[4] as shown in Figure 19c. Further, Figure 19d shows the tail FCT breakdown for flows with different flow sizes under 1:1 over-subscription network and load of 0.7. We omit other scenarios since their FCT breakdown results are similar.

F. Sensitivity Analysis

Path migration freeze window: Figure 20a and Figure 20b shows insights on how vFab's path migration freeze window impacts the convergence speed. Under light workload (50%), the whole network can converge within 500μs regardless of the freeze window parameter. As the average load goes up to 70%, vFab's slow migration policy shows its advantage especially in decreasing path migration frequency. We select

[4]FCT slowdown means the actual FCT normalized by the expected FCT, *i.e.*, dividing flow size by VM's minimum bandwidth demand in Hose model.

the waiting time as [1, 10] RTTs, which keeps the convergence time low and significantly reduces the oscillation risk.

Probing frequency: We randomly generate background traffic of 50% load, and then select 16 senders and 1 receiver to generate incast traffic. Figure 20c shows that with periodic probing, *e.g.*, probe packets are sent every n RTTs, vFab has similar convergence times with the default probing scheme, because the stale information captured by lazy probing could lead to an aggressive reaction in each rate control loop, which makes the rate converge in fewer loops.

VI. Related Work

The dependable virtual fabric has been an enduring research topic. vFab is the first that leverages the informative data plane to ensure VF dependability in bandwidth guarantee, work conservation, bounded latency, and resilient reachability simultaneously.

Bandwidth : Static bandwidth reservation [31], [32], [33], [34] provides strong guarantees but with low resource utilization. FairCloud (PS-P) [18] and NetShare [36] require per VM/tenant queues for bandwidth allocation, which is infeasible in commodity switches; Seawall [35] uses TCP-like algorithms to dynamically allocate bandwidth. FairCloud (PS-L/N) [18] and ElasticSwitch [15] both adopt Seawall as their fabric solutions, while ElasticSwitch enables tenant-level bandwidth policy rather than Seawall's per-source, which is important to be tenant strategy-free [18]. Edge-based solutions, *e.g.*, PicNIC [17], EyeQ [16] and GateKeeper [40], use end-to-end admission control to dynamically assign bandwidth for minimum guarantee and work conserving. However, they require a congestion/loss-free fabric that is not practical.

Latency : Many solutions achieve low latency using queue reduction [45], [46], [64], traffic prioritization [65], [66], deadline-aware scheduling [67], [68], [69], in dedicated networks. However, those works do not directly provide bounded latency in multi-tenant DCNs. QJUMP [29] uses priority queuing and rate-limiting, Silo [34] uses network calculus, and Chameleon [30] further exploits path diversity to bounded packet delay. All above approaches are static allocation and lack of work conservation. Based on ElasticSwitch, Trinity [44] leverages priority queues to prioritize the traffic of bandwidth guarantee over that of work conservation, thereby achieving low latency for short flows. By contrast, vFab limits the burst size to achieve both bounded latency and work conservation simultaneously, with a single queue.

Reachability : FRR [23], F10 [24], DDC [25], and SharedBackup [26] exploit local path redundancy at switches for fast rerouting upon failure. MPTCP [27] and MPQUIC [28] distribute a single flow to multiple sub-flows to provide resilience against network failures. Compared to those schemes, vFab relies on edge to explicitly migrate paths upon failure within sub-millisecond.

Informative core : Existing works leverage the network information from programmable switches for many aspects, such as HPCC [45] for congestion control, Clove [22] for load balancing, NetSeer [42] and BufScope [70] for monitoring, *etc.* By contrast, vFab is the first that defines the critical information for providing dependable fabric service and builds a dynamic-path framework with active edge and informative core fusion.

Working with load balancing : Numerous works [20], [22], [71], [72] aim to improve the overall network throughput and fault tolerance with better load balancing. vFab works with any per-flow load balancing solutions, *e.g.*, ECMP, since they are mostly common in today's DCNs. We leave how to work with other solutions, *e.g.*, per-packet [73], per-flowlet [20], [74] or per-flowcell [75] load balancing, as our future work.

VII. Conclusion

We believe the newly available programmable data plane is the key to solving the exceptional challenges of providing dependable fabric in multi-tenant DCNs – vFab is such an example. vFab constructs an dependable virtualized fabric service via a fusion of an active edge and an informative core. Its innovation lies in the simple and effective mechanisms to make the whole network converge to the dependable tenant-level performance (*e.g.*, resilient reachability, guaranteed bandwidth and bounded latency) and high utilization in sub-millisecond timescale. We demonstrate that vFab can be efficiently implemented with commodity SmartNICs and programmable switches.

Appendix A
The Analysis of vFab's Theoretical Properties

In this Section we review some of the theoretical background to fairness and to convergence properties of dual congestion control algorithms.

A. Fairness

Suppose there are resources $i = 1, 2, \cdots, I$ and paths $j = 1, 2, \cdots, J$. Let A be the incidence matrix containing only zeroes and ones defined by $A_{ij} = 1$ if resource i is used by path j and $A_{ij} = 0$ otherwise; assume each path uses at least one resource, so that no column of A is identically zero. Let $C_i > 0$ be the (target) capacity of resource i, for $i = 1, 2, \cdots, I$, and define the vector $C = (C_i, i = 1, 2, \cdots, I)$. A rate allocation is a vector $x = (x_j, j = 1, 2, \cdots, J)$. Let y_i be the load on resource i and let $y = (y_i, i = 1, 2, \cdots, I)$. From the definition of the matrix A we have that $y = Ax$. Note that the vector inequality $y = Ax \leq C$ is satisfied if and only if the load on each resource is less than or equal to the (target) capacity of the resource.

Let $w_j > 0, j = 1, 2, \cdots, J$, be a set of weights, and let $\alpha \in (0, \infty)$ be a fixed constant. The *weighted α-fair allocation* $x = (x_j, j = 1, 2, \cdots, J)$ is, for $\alpha \neq 1$, the solution of the following optimization problem

$$\text{maximize} \sum_j \frac{w_j}{1-\alpha} \left(\frac{x_j}{w_j}\right)^{1-\alpha} \quad \text{over } x \geq 0, Ax \leq C. \tag{5}$$

For $\alpha = 1$ it is the solution of the same optimization problem but with objective function $\sum_j w_j \log(x_j/w_j)$, the natural continuation of the objective function through $\alpha = 1$ [76], [77].

The solution to this optimization problem takes the form

$$x_j = w_j \left(\sum_i A_{ij} R_i^{-\alpha}\right)^{-1/\alpha} \tag{6}$$

where $R = (R_i, i = 1, 2, \cdots, I)$ are a set of link rates: R_i is the rate which would be allocated to a source of unit weight whose path went through just one resource, resource i.

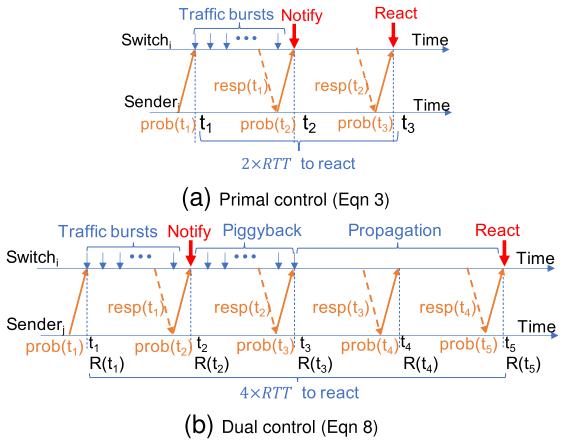


Fig. 21. Illustration of vFab's convergence.

Note that as $\alpha \to \infty$ the expression (6) approaches

$$x_j = w_j \min_{i:A_{ij}=1} \{R_i\}$$

corresponding to weighted max-min fairness as used in §III. The case $\alpha = 1$ corresponds to weighted proportional fairness. If $w \equiv 1$ then as $\alpha \to 0$ the expression (6) approaches the rate allocation which maximizes the sum of rates over all paths [76].

Max-min is the fairness criterion most commonly discussed for communication networks, but it can place too much emphasis on fairness over efficiency. Proportional fairness has preferable efficiency properties in networks with routing choices and heterogeneous loadings [78].

B. Equilibrium Rates and Convergence

Suppose now the link rates $R(n)$ at time n are updated in discrete time by the recursion

$$y(n) = Ax(n) \tag{7}$$

$$R_i(n+1) = R_i(n)\frac{C_i}{y_i(n)} \tag{8}$$

where the sending rates x are given by the expression (6). In discrete time, with exactly one RTT needed to update the sending rates, this corresponds to the update rule in §III-C. An equilibrium point of the recursion (7)-(8) is exactly a solution to the optimization problem (5).

Figure 21 illustrates the convergence of the primal control and dual control. It only provides a rough intuition and should not be considered a formal proof. The actual convergence delay in both cases may vary due to the random arrivals, actual RTT variance, etc. Figure 21a shows that the primal control of Eqn 3 takes 2 RTTs to let the senders to learn the burst traffic demands and thus the peak latency spike is roughly bounded within the double of the maximum base RTT in the network. Figure 21b shows that the dual control of Eqn 8 takes about 4 RTTs to converge. It spends first RTT to notice the traffic bursts by computing $R(t_2)$. Then, it waits for the second RTT for senders to piggyback $R(t_2)$ as their sending rate. After the propagation delay of 1-2 RTTs, the switch receives all the traffic amount dictated by $R(t_2)$ and computes the next rate $R(t_5)$ accordingly.

The difficulty with the recursion (7)-(8) is that its convergence is very sensitive to RTTs. The stability of similar algorithms has been investigated by several authors using a variety of simplified models [79], [80], [81], [82]. The main insights we draw from previous work are firstly that the rate of adaptation should be scaled to round-trip times in order to avoid destabilising oscillations, and secondly that while feedback based on queue size is important for dealing with transient overloads, it is not helpful for steady state stability when we already have feedback based on rate mismatch. (Observe that the queue size is simply the total rate mismatch over the random period since the queue was last empty, and thus does not give information additional to rate mismatch.)

In HPCC [45], convergence of utilization was fast but convergence to fairness was slower, since it needed to be effected by an AIMD scheme implemented at sources. In the approach of this paper, the parameters $R = (R_i, i = 1, 2, \cdots, I)$, or their scaled versions, one for each resource, allow fast convergence for both utilization and fairness. In the next Section we show the importance of these parameters, and how they can in principle be computed at the sources from measurements made at resources and conveyed back to sources.

C. Impact of Delayed Feedback

To understand the impact of delayed feedback on stability we review existing theoretical results.

For each i, j such that $A_{ij} = 1$, let T_{ji} be the delay from the source of flow on path j to the resource i, and let T_{ij} be the return delay from resource i back to the source. Then

$$T_{ji} + T_{ij} = T_j \tag{9}$$

where T_j is the round-trip delay on path j. Consider the continuous time model

$$\frac{d}{dt} R_i(t) = \kappa \frac{R_i(t)}{\bar{T}_i}\left(1 - \frac{y_i(t)}{C_i}\right) \tag{10}$$

where

$$y_i(t) = \sum_j A_{ij} x_j(t - T_{ji}) \tag{11}$$

is the aggregate load at resource i, the rate x_j is given by

$$x_j(t) = w_j \left(\sum_i A_{ij} R_i(t - T_{ij})^{-\alpha}\right)^{-1/\alpha} \tag{12}$$

and

$$\bar{T}_i = \frac{\sum_j A_{ij} x_j(t) T_j}{\sum_j A_{ij} x_j(t)} \tag{13}$$

is the average round-trip delay over all packets passing through resource i. The equilibrium of the dynamical system (10)-(12) is the weighted α-fair allocation and is locally stable [81] if

$$\kappa < \frac{\pi}{2}.$$

Important points to note are that the stability result does not depend on α, i.e. which fairness criterion is used, nor on the heterogeneity of round-trip delays, nor on the topology of the network. The scaling in expression (10) shows that the speed of adaptation of R_i should be inversely related to the average round-trip delay $\bar{T}_i$. We do not need to know $\bar{T}_i$ precisely, we simply need to ensure that we can bound it above since the condition on κ is an inequality.

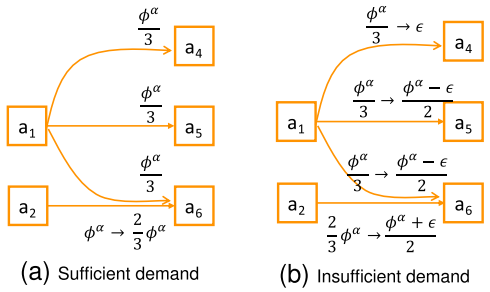


Fig. 22. Example of vFab's token assignment.

APPENDIX B
TOKEN ASSIGNMENT

We present the algorithm to partition a VF's minimum bandwdith gurantee into VM-to-VM guarantees under online traffic patterns. The algorithm solves this problem via pairwise token assignment. The actual traffic is either bounded by the sender's or the receiver's capacity. We have in total ϕ^α tokens in both sides for minimum bandwidth guarantee B^α_{min} at the VF α, where we have $\phi^\alpha = \frac{B^\alpha_{min}}{B_u}$ and B_u is predefined bandwidth for a unit token. On the one hand, the sender apportions tokens across VM-pairs to fully utilize the bandwidth and conveys the allocated token as "demand" to the receiver. The receiver, on the other hand, responds the demand using max-min fair sharing. We explain the details with an example in the following.

Figure 22a shows an example of token assignment algorithm with four VM-pairs. Initially, each sider equally distributes ϕ^α for all active VM-pairs, so that $\phi_{a_1 \to a_4} = \phi_{a_1 \to a_5} = \phi_{a_1 \to a_6} = \frac{\phi^\alpha}{3}$ and $\phi_{a_2 \to a_6} = \phi^\alpha$. The receiver uses max-min fairing to arbitrate the incoming token demands with responses. For example, a_6 reponds $\phi_{a_1 \to a_6} = \frac{\phi^\alpha}{3}$ and $\phi_{a_2 \to a_6} = \frac{2}{3}\phi^\alpha$ to a_1 and a_2 respectively.

Right now, we only talk about assignment with sufficient traffic demand. Figure 22b shows the case where VM-pair $a_1 \to a_4$ has insufficient traffic demand of ϵ. Each sender keeps monitoring the actual TX rate for each VM-pair. When a VM-pair's actual TX rate is less than its assigned token in last epoch, the sender redistributes the spare tokens to other VM-pairs for work conserving. As the case in Figure 22b, a_1 redistributes the spare capacity to the remaining VM-pairs so that $\phi_{a_1 \to a_5} = \phi_{a_1 \to a_6} = \frac{\phi^\alpha - \epsilon}{2}$. After one iteration, VM-pair $a_2 \to a_6$ would adapt to $\phi_{a_2 \to a_6} = \frac{\phi^\alpha + \epsilon}{2}$.

Our framework supports several options to redistribute the tokens from VM-pair of insufficient demand. ElasticSwitch [15] uses the measured demand as the guarantee and, once satisfied, exponetially increases its guarantee to reach fair-sharing. Here we propose another option. We assign at least the fair-sharing token to each VM-pair even if its demand is insufficient (Line 7 in Algorithm 1). The key point is that it gives the VM-pair a rapid way to grow its bandwidth when it has immediate traffic demand. In the worst case, we only assign double the VM-pair's token to the network in one RTT. Thus, the bandwidth and latency are still bounded and they should converge very fast. The full algorithm is presented in Algorithm 1.

APPENDIX C
SUPPORT MULTIPLE UNDERLAY PATHS PER VM-PAIR

vFab supports to use multiple underlay paths for each VM-pair and allows tenant applications to spread traffic over them. We ensure that VM-pair token assigned from Algorithm 1 is properly distributed onto paths, while ensuring fairness and work conserving properties.

Algorithm 2 shows the overall procedure. To ensure fairness, we split the total VM-pair token equally to all paths initially (Line 3). In case some path has insufficient traffic demand, we dynamically redistribute the spare capaity to other paths for work conserving (Line 11). Nevertheless, we admit each path at least a fair-sharing token even if it has insufficient traffic demand (Line 7). This approach boosts traffic demand growth with slightly more traffic in one RTT.

Algorithm 1 Token Assignment Algorithm. ϕ^α Is the Hose Model Weight of VF α. $\boldsymbol{P}$ Is a Group of VM-Pairs Either From Sender or Towards Receiver; Each VM-Pair Has Three Fields: tx Is Its Actual TX Rate, ϕ_S Is Sender Assigned Token, and ϕ_D Is Receiver Admitted Tokens

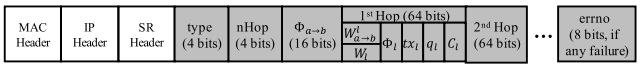
1: **procedure** TOKENASSIGNMENT(ϕ^α, $\boldsymbol{P}$) ▷ Sender
2: $Y = 0$, $N' = 0$, $N_S = |\boldsymbol{P}|$, $\forall p$ in $\boldsymbol{P}, p.\phi_S = 0$;
3: $\bar{\phi} = \frac{\phi^\alpha}{N_S}$;
4: **for** each VM-pair p in $\boldsymbol{P}$ **do**
5: **if** $\bar{\phi} > \frac{p.tx}{B_u}$ **then**
6: $Y += (\bar{\phi} - \frac{p.tx}{B_u})$;
7: $p.\phi_S = \frac{p.tx}{B_u}$; ▷ Bouned by demand but sender still admits $\bar{\phi}$.
8: $N' += 1$;
9: $\bar{\phi} += \frac{Y}{N_S - N'}$;
10: $\boldsymbol{P}^s = Sort(\boldsymbol{P})$; ▷ Ascending on ϕ_D
11: **for** each VM-pair p in $\boldsymbol{P}^s$ **do**
12: **if** $p.\phi_D < \bar{\phi}$ and $p.\phi_S = 0$ **then**
13: $N' += 1$;
14: $\bar{\phi} += (\frac{\bar{\phi} - p.\phi_D}{N_S - N'})$;
15: $p.\phi_S = p.\phi_D$; ▷ Bounded by receiver
16: **for** each VM-pair p in $\boldsymbol{P}$ **do**
17: **if** $p.\phi_S = 0$ **then**
18: $p.\phi_S = \bar{\phi}$; ▷ Redistribute unused tokens.
return $\boldsymbol{P}$;
19: **procedure** TOKENADMISSION(ϕ^α, $\boldsymbol{P}$) ▷ Receiver
20: $N' = 0$, $N_D = |\boldsymbol{P}|$;
21: $\bar{\phi} = \frac{\phi^\alpha}{N_D}$;
22: $\boldsymbol{P}^s = Sort(\boldsymbol{P})$; ▷ Ascending on ϕ_S
23: **for** each VM-pair p in $\boldsymbol{P}$ **do**
24: **if** $p.\phi_S < \bar{\phi}$ **then**
25: $p.\phi_D = UNBOUND$;
26: $N' += 1$;
27: $\bar{\phi} += \frac{\bar{\phi} - p.\phi_S}{N_D - N'}$; ▷ Redistribute unused tokens.
28: **else**
29: $p.\phi_D = \bar{\phi}$;
30: **return** $\boldsymbol{P}$;

APPENDIX D
PACKET FORMAT

Figure 23 shows the probe/response packet format of vFab. The IP header is reserved so that the switch can use IP-based routes when bouncing probe. We use customized SR header to enforce source routing, followed by our customized INT header. Note that we can use standard SR-MPLS [83], but

Algorithm 2 Multipath Token Assignment Algorithm. ϕ_S Is Sender Assigned Token for the VM-Pair. $\boldsymbol{L}$ Is a Group of Paths Used by the VM-Pair; Each Path in $\boldsymbol{L}$ has Two Fields: Actual TX Rate tx and Token ϕ to Be Assigned

procedure MULTIPATHASSIGNMENT(ϕ_S, $\boldsymbol{L}$)
  $Y = 0$, $N' = 0$, $N_L = |\boldsymbol{L}|$, $\forall l$ in $\boldsymbol{L}, l.\phi = 0$;
  $\bar{\phi} = \frac{\phi_S}{N_L}$; ▷ Ensure fairness.
  for each path l in $\boldsymbol{L}$ **do**
    if $\bar{\phi} > \frac{l.tx}{B_u}$ **then**
      $Y += (\bar{\phi} - \frac{l.tx}{B_u})$;
      $l.\phi = \bar{\phi}$; ▷ Boost demand growth.
      $N' += 1$;
  for each path l in $\boldsymbol{L}$ **do**
    if $l.\phi = 0$; **then**
      $l.\phi = \bar{\phi} + \frac{Y}{N_L - N'}$; ▷ Redistribute unused token.
  return $\boldsymbol{L}$;



type (4 bits) : the type of the packet. 1 for probe, 2 for response, 4 for failure response.
nHop (4 bits) : the number of switches the packet has passed through.
$\Phi_{a\to b}$ (16 bits) : the sender (or receiver) bandwidth token allocated for VM-pair $a \to b$, if type is 1 (or 2).
$W^l_{a\to b}$ (16 bits) : the sending window of VM-pair $a \to b$ traversing link l. It is replaced by W_l after the switch inserts VF information.
W_l (16 bits) : the total sending window of all VM-pairs traversing link l.
Φ_l (16 bits) : the total token of all active VM-pairs on l.
tx_l (16 bits) : the actual TX rate of link l.
q_l (12 bits) : the real time queue size of link l.
C_l (4 bits) : the type of speed of the egress port, e.g., 10Gbps, 100Gbps, etc.
errno (8 bits) : the type of failure, if failure happens, i.e., if type is 4; otherwise, it is not required.

Fig. 23. The probe/response packet format of vFab.

to reduce the length of the packet header, we simplify the SR header. Similarly, we use the customized INT header to reduce the bandwidth overhead of probe. The field *type* is used to distinguish between probe packet, response packet and failure notification packet. The field *nHop* represents the number of hops passed by the data packet, and also indicates how many switches' INT information is contained in the following fields. The field $\phi_{a\to b}$ is the bandwidth token allocated for VM-pair $a \to b$ by end host. For probe, it is the token assigned by sender; for response, it is the one assigned by receiver. The INT information of each hop consists of 5 fields, *i.e.*, $W^l_{a\to b}$ (W_l), ϕ_l, tx_l, q_l and C_l. $W^l_{a\to b}$ and W_l share the same field, where the former carries the sending window of VM-pair $a \to b$ traversing link l, and after it is read by switches, the switch inserts the aggregated sending window information represented by W_l. The field ϕ_l is the total active tokens maintained by switch's Bloom Filter. The last three fields represent the network load status and network capacity. If vFab-C detects network failure, it will insert *errno* field to indicate which kind of failure happens in the network.

REFERENCES

[1] S. Wang et al., "Predictable vFabric on informative data plane," in *Proc. ACM SIGCOMM*, 2022, pp. 615–632.
[2] (2020). *Alibaba Cloud EBS Performance*. [Online]. Available: https://www.alibabacloud.com/help/doc-detail/25382.htm
[3] (2020). *Google Cloud Block Storage Performance*. [Online]. Available: https://cloud.google.com/compute/docs/disks/performance
[4] (2020). *Amazon EBS Features*. [Online]. Available: https://aws.amazon.com/ebs/features/
[5] (2020). *Microsoft Azure Disk Storage*. [Online]. Available: https://azure.microsoft.com/en-us/services/storage/disks/
[6] Y. Gao et al., "When cloud storage meets RDMA," in *Proc. USENIX NSDI*, 2021, pp. 519–533.
[7] V. J. Reddi et al., "MLPerf inference benchmark," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2020, pp. 446–459.
[8] S. Rajbhandari et al., "DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale," 2022, *arXiv:2201.05596*.
[9] B. Arzani et al., "007: Democratically finding the cause of packet drops," in *Proc. USENIX NSDI*, 2018, pp. 419–435.
[10] A. Roy, H. Zeng, J. Bagga, and A. C. Snoeren, "Passive realtime datacenter fault detection and localization," in *Proc. USENIX NSDI*, 2017, pp. 595–612.
[11] P. Lapukhov, A. Premji, and J. Mitchell, "Use of BGP for routing in large-scale data centers," Internet Eng. Task Force, Tech. Rep., RFC 7938, 2016.
[12] A. Singh et al., "Jupiter rising: A decade of CLOS topologies and centralized control in google's datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 183–197, 2015.
[13] R. B. Da Silva and E. S. Mota, "A survey on approaches to reduce BGP interdomain routing convergence delay on the internet," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2949–2984, 4th Quart., 2017.
[14] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 266–277, 2011.
[15] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing," in *Proc. ACM SIGCOMM Conf.*, Aug. 2013, pp. 351–362.
[16] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, A. Greenberg, and C. Kim, "EyeQ: Practical network performance isolation at the edge," in *Proc. USENIX NSDI*, 2013, pp. 297–311.
[17] P. Kumar et al., "PicNIC: Predictable virtualized NIC," in *Proc. ACM SIGCOMM*, 2019, pp. 351–366.
[18] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the network in cloud computing," in *ACM SIGCOMM*, 2012, pp. 187–198.
[19] C. Hopps et al., "Analysis of an equal-cost multi-path algorithm," Internet Eng. Task Force, Tech. Rep., 2992, Nov. 2000.
[20] M. Alizadeh et al., "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM SIGCOMM*, 2014, pp. 503–514.
[21] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proc. SOSR*, 2016, pp. 1–12.
[22] N. Katta et al., "Clove: Congestion-aware load balancing at the virtual edge," in *Proc. ACM CoNEXT*, 2017, pp. 323–335.
[23] G. Enyedi, A. Csaszar, A. Atlas, C. Bowers, and A. Gopalan, "An algorithm for computing IP/LDP fast reroute using maximally redundant trees (MRT-FRR)," Internet Eng. Task Force, Tech. Rep., 7811, p. 118, 2016.
[24] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *Proc. USENIX NSDI*, 2013, pp. 399–412.
[25] J. Liu, A. Panda, A. Singla, B. Godfrey, M. Schapira, and S. Shenker, "Ensuring connectivity via data plane mechanisms," in *Proc. USENIX NSDI*, 2013, pp. 113–126.
[26] D. Wu, Y. Xia, X. S. Sun, X. S. Huang, S. Dzinamarira, and T. S. E. Ng, "Masking failures from application performance in data center networks with shareable backup," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 176–190.
[27] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. USENIX NSDI*, 2011, pp. 1–14.
[28] Q. D. Coninck and O. Bonaventure, "Multipath QUIC: Design and evaluation," in *Proc. ACM CoNEXT*, 2017, pp. 160–166.
[29] M. P. Grosvenor et al., "Queues don't matter when you can JUMP them!" in *Proc. USENIX NSDI*, 2015, pp. 1–14.
[30] A. V. Bemten et al., "Chameleon: Predictable latency and high utilization with queue-aware and adaptive source routing," in *Proc. ACM CoNEXT*, 2020, pp. 451–465.
[31] C. Guo et al., "SecondNet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. ACM CoNEXT*, 2010, pp. 1–12.
[32] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 242–253.

[33] J. Lee et al., "Cloudmirror: Application-aware bandwidth reservations in the cloud," in *Proc. USENIX HotCloud*, 2013, pp. 1–6.

[34] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable message latency in the cloud," in *Proc. ACM SIGCOMM*, 2015, pp. 435–448.

[35] A. Shieh, S. Kandula, A. G. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. USENIX NSDI*, 2011, pp. 1–14.

[36] V. T. Lam, S. Radhakrishnan, R. Pan, A. Vahdat, and G. Varghese, "Netshare and stochastic netshare: Predictable bandwidth allocation for data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 3, pp. 5–11, 2012.

[37] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *Proc. USENIX NSDI*, 2013, pp. 171–184.

[38] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: Incorporating time-varying network reservations in data centers," in *Proc. ACM SIGCOMM*, 2012, pp. 199–210.

[39] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "HUG: Multi-resource fairness for correlated and elastic demands," in *Proc. USENIX NSDI*, 2016, pp. 407–424.

[40] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. O. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proc. WIOV*, 2011, pp. 1–8.

[41] Z. Zhang et al., "Hashing linearity enables relative path control in data centers," in *Proc. USENIX ATC*, 2021, pp. 855–862.

[42] Y. Zhou et al., "Flow event telemetry on programmable data plane," in *Proc. ACM SIGCOMM*, 2020, pp. 76–89.

[43] X. Wu et al., "NetPilot: Automating datacenter network failure mitigation," in *Proc. ACM SIGCOMM*, 2012, pp. 419–430.

[44] S. Hu, W. Bai, K. Chen, C. Tian, Y. Zhang, and H. Wu, "Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud," *IEEE Trans. Cloud Comput.*, vol. 9, no. 2, pp. 763–776, Apr. 2021.

[45] Y. Li et al., "HPCC: High precision congestion control," in *Proc. ACM SIGCOMM*, 2019, pp. 44–58.

[46] G. Kumar et al., "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. ACM SIGCOMM*, 2020, pp. 514–528.

[47] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, 1988.

[48] L. Zhang and Y. Guan, "Detecting click fraud in pay-per-click streams of online advertising networks," in *Proc. 28th Int. Conf. Distrib. Comput. Syst.*, Jun. 2008, pp. 77–84.

[49] Y. Li, R. Miao, C. Kim, and M. Yu, "LossRadar: Fast detection of lost packets in data center networks," in *Proc. 12th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2016, pp. 481–495.

[50] (2019). *In-band Network Telemetry in Broadcom Trident3*. [Online]. Available: https://www.broadcom.com/blog/new-trident-3-switch-delivers-smarterprogrammability-for-enterprise-and-service-provider-datacenters

[51] (2019). *In-band Network Telemetry in Barefoot Tofino*. [Online]. Available: https://www.opencompute.org/files/INT-In-Band-Network-Telemetry-APowerful-Analytics-Framework-for-your-Data-Center-OCP-Final3.pdf

[52] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[53] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 5–12, 2005.

[54] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the internet," in *Proc. IWQos*. Cham, Switzerland: Springer, 2005, pp. 271–285.

[55] *Infiniband Architecture Specification Volume 1 Release 1.3*, InfiniBand Trade Assoc., 2015.

[56] R. Joshi, B. Leong, and M. C. Chan, "TimerTasks: Towards time-driven execution in programmable dataplanes," in *Proc. ACM SIGCOMM Conf.*, 2019, pp. 69–71.

[57] M. Kogias, M. Weber, and E. Bugnion, "SLOG: Your switch is also your load-generator," in *Proc. USENIX NSDI Poster*, 2020, pp. 1–4.

[58] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "BurstRadar: Practical real-time microburst monitoring for datacenter networks," in *Proc. 9th Asia–Pacific Workshop Syst.*, Aug. 2018, pp. 1–8.

[59] (2021). *P4 Language Consortium, Portable Switch Architecture*. [Online]. Available: https://p4.org/p4-spec/docs/PSA.html

[60] *NS3 Simluator, NS3*. Accessed: Feb. 10, 2022. [Online]. Available: https://www.nsnam.org/

[61] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Oct. 2008.

[62] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *Proc. ACM SIGMETRICS*, 2012, pp. 53–64.

[63] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 431–442, Sep. 2012.

[64] M. Alizadeh et al., "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM*, 2010, pp. 63–74.

[65] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "HOMA: A receiver-driven low-latency transport protocol using network priorities," in *Proc. ACM SIGCOMM*, 2018, pp. 221–235.

[66] M. Alizadeh et al., "PFABRIC: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, 2013.

[67] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, Sep. 2012.

[68] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 50–61, 2011.

[69] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 127–138, Sep. 2012.

[70] K. Gao et al., "Buffer-based end-to-end request event monitoring in the cloud," in *Proc. USENIX NSDI*, 2022, pp. 829–843.

[71] G. Chen et al., "Fast and cautious: Leveraging multi-path diversity for transport loss recovery in data centers," in *Proc. USENIX ATC*, 2016, pp. 29–42.

[72] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *Proc. ACM SIGCOMM*, 2017, pp. 225–238.

[73] J. Cao et al., "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proc. ACM CoNEXT*, 2013, pp. 49–60.

[74] E. Vanini, R. Pan, M. Alizadeh, and P. Taheri, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *Proc. USENIX NSDI*, 2017, pp. 407–420.

[75] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 465–478, 2015.

[76] F. Kelly and E. Yudovina, *Stochastic Networks*. Cambridge, U.K.: Cambridge Univ. Press, 2014.

[77] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 556–567, Oct. 2000.

[78] B. Radunovic and J. Y. Le Boudec, "Rate performance objectives of multihop wireless networks," *IEEE Trans. Mobile Comput.*, vol. 3, no. 4, pp. 334–349, Oct. 2004.

[79] H. Balakrishnan, N. Dukkipati, N. Mckeown, and C. Tomlin, "Stability analysis of explicit congestion control protocols," *IEEE Commun. Lett.*, vol. 11, no. 10, pp. 823–825, Oct. 2007.

[80] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2002, pp. 89–102.

[81] F. Kelly, G. Raina, and T. Voice, "Stability and fairness of explicit congestion control with small buffers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 51–62, Jul. 2008.

[82] T. Voice and G. Raina, "Stability analysis of a max–min fair rate control protocol (RCP) in a small buffer regime," *IEEE Trans. Autom. Control*, vol. 54, no. 8, pp. 1908–1913, Aug. 2009.

[83] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.