Predictable vFabric on Informative Data Plane

Shuai Wang[⋆‡†], Kaihui Gao[⋆†], Kun Qian[†], Dan Li[⋆‡], Rui Miao[†], Bo Li[†], Yu Zhou[†], Ennan Zhai[†], Chen Sun[†], Jiaqi Gao[†], Dai Zhang[†], Binzhang Fu[†], Frank Kelly[◇], Dennis Cai[†], Hongqiang Harry Liu[†], Ming Zhang[†]
[⋆]*Tsinghua University* [†]*Alibaba Group* [‡]*Zhongguancun Laboratory* [◇]*University of Cambridge*

ABSTRACT

In multi-tenant data centers, each tenant desires reassuring predictability from the virtual network fabric – *bandwidth guarantee*, *work conservation*, and *bounded tail latency*. Achieving these goals simultaneously relies on rapid and precise traffic admission. However, the slow convergence (tens of milliseconds) of prior works can hardly satisfy the increasingly rigorous performance demand under dynamic traffic patterns. Further, state-of-the-art load balance schemes are all guarantee-agnostic and bring great risks on breaking bandwidth guarantee, which is overlooked in prior works.

In this paper, we propose μFAB, a predictable virtual fabric solution which can (1) explicitly select proper paths for all flows and (2) converge to ideal bandwidth allocation at sub-millisecond timescales. The core idea of μFAB 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 μFAB with commodity SmartNICs and programmable switches. Evaluations show that μFAB can keep minimum bandwidth guarantee with high bandwidth utilization and near-optimal transmission latency in various network situations with limited probing bandwidth overhead. Application-level experiments, *e.g.*, compute and storage scenarios, show that μFAB can improve QPS by 2.5× and cut tail latency by more than 21× compared to the alternatives.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **Transport protocols**; **Cloud computing**;

KEYWORDS

Programmable Data Plane, Performance Isolation

ACM Reference Format:
Shuai Wang, Kaihui Gao, Kun Qian, Dan Li, Rui Miao, Bo Li, Yu Zhou, Ennan Zhai, Chen Sun, Jiaqi Gao, Dai Zhang, Binzhang Fu, Frank Kelly, Dennis Cai, Hongqiang Harry Liu, Ming Zhang. 2022. Predictable vFabric on Informative Data Plane. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3544216.3544241

Shuai Wang and Kaihui Gao are the co-primary authors.

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08.
https://doi.org/10.1145/3544216.3544241

1 INTRODUCTION

In multi-tenant data centers, virtual machines (VMs)[1] of a tenant are expected to be logically interconnected by a virtual network fabric (VF) as if in a dedicated cluster, even though all tenants share the same physical network. While many solutions [12, 21, 29, 37, 51] have been proposed to improve the performance of multi-tenant data center networks (DCNs), they are not competent to provide a strongly predictable VF service – *bandwidth guarantee*, *work conservation*, and *bounded tail latency* for the following two reasons.

First, the convergence speed (tens of milliseconds) of prior predictable VF works fails to catch up with the increasingly rigorous performance demand from today's applications[2]. Multiple factors drive this trend. On the one hand, resource pooling is an inevitable trend in data centers with strict resource access deadlines [2–5]. For instance, being the highest performance level disk in Elastic Block Storage, the enhanced SSD requires the I/O operation latency to be 100μs on average and 1ms at tails [18, 57]. On the other hand, with the emerging specialized compute accelerators, the performance bottleneck of distributed computing applications (*e.g.*, distributed machine learning) is shifting from computation to communication [50, 58]; thus, they require instantaneously available bandwidth every time the parameter/activation transfer starts, especially for distributed machine learning inference that typically involves multiple transfers and needs to respond to online queries within 10ms [47, 48]. Hence, handling traffic dynamics rapidly, *i.e.*, at sub-millisecond timescales, is critical to meet the performance demands of today's applications.

Second, end-to-end bandwidth guarantees could be easily broken by guarantee-agnostic path management schemes. Existing solutions [29, 37, 44, 45] providing bandwidth guarantee with work conservation mostly view the network fabric as an aggregated pipe between source and destination, assuming that specific path-selections are made by complementary load balancing schemes, *e.g.*, selecting a random path [24] or the least-utilized path [7, 31, 32]. However,*link utilization* (actual traffic) and *link subscription* (traffic with bandwidth guarantee) are not equivalent due to the work conservation. When a new flow with a high traffic demand enters the network, assigning it to the least utilized path may violate bandwidth guarantees of others. Thus, all flows on this path converge to new rates lower than their bandwidth guarantee and face significant performance degradation (a detailed example is shown in § 2.2). Note that guaranteeing the minimum bandwidth is mandatory, but providing extra capacity is a bonus. Hence, subscription-aware path selection is essential to provide a predictable VF.

Fundamentally, to make rapid VF convergence and correct path selection, we observe that the key is the fine-grained network status, *e.g.*, bandwidth subscription, and link utilization. However, due to the

[1] Or other forms of compute virtualization.
[2] Making a VF converge means that the system simultaneously meets all three goals – bandwidth guarantee, work conservation and bounded tail latency.

lack of fine-grained network status, prior works have to suffer from heuristic rate adjustment and utilization-oriented load balance, even random path selection. 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 predictability framework.

In this paper, we propose μFAB, a framework that provides a strongly predictable VF service to data center tenants. By leveraging programmable NICs and switches in modern data centers, μFAB simultaneously provides minimum bandwidth guarantee, work conservation, and bounded tail latency in an end-to-end way. Specifically, μFAB 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 utilization and active bandwidth subscription, back to the edge via In-band Network Telemetry (INT). 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 provides a foundation for rapid detection and mitigation of performance degradation.

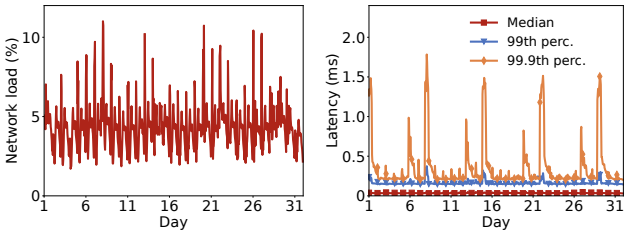
There are three main challenges to realizing μFAB. First, since the data center traffic mix changes swiftly, it is necessary to quickly provision bandwidth once a tenant has traffic demand while efficiently allowing other tenants to share the reserved but unused bandwidth. Second, data center traffic is bursty in nature, *e.g.*, incast, which requires the distributed framework to admit traffic cooperatively across hosts to avoid traffic interference among tenants at short timescales. Third, each edge reacts to path quality changes independently, resulting in a prolonged convergence process and traffic oscillation.

μFAB addresses these challenges with three innovations:

(i) *Hierarchical bandwidth allocation.* First, the edge selects a path for each flow to keep that the total active bandwidth subscription, *i.e.*, the sum of minimum bandwidth guarantees of tenants that pass through the link, does not exceed the link capacity. Hence, the minimum bandwidth can be guaranteed for all tenants, if the link capacity is shared by the flows proportionally to their minimum bandwidth. Then, the edge swiftly and accurately adjusts sending rate to make the bandwidth utilization converge to the target. Therefore, even if some tenants have insufficient demands, the unused bandwidth can be quickly utilized by other tenants that share the same link; conversely, if a tenant has an immediate traffic demand, it can rapidly grab its guaranteed bandwidth back. Our theoretical analysis suggests that μFAB can achieve both strictly guaranteed minimum bandwidth and high network utilization (§ 3.3);

(ii) *Two-stage and window-based traffic admission.* In order to avoid queuing, each edge uses a window updated by bandwidth utilization,*i.e.*, utilization-based window, to limit a tenant's inflight traffic on a path. Across hosts, μFAB controls each tenant's total burst up to their minimum bandwidth guarantee and additively increases their sending windows until utilization-based windows ramp down, and starts to use the latter. Thus, μFAB can bound the queue size on bottleneck link to three times of BDP (Bandwidth-Delay Product) (§ 3.4);

(iii) *Accurate and stable path migration.* μFAB makes a timely and accurate judgment on available bandwidth and the risk of latency spikes on a path with a single probe instead of indeed putting traffic on the path. 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; μFAB's path migration can also avoid oscillations and packet re-ordering (§ 3.5).



(a) Network load (b) RTT

Figure 1: Bursty traffic interference in ECS scenario.

We implement μFAB in commodity SmartNICs and programmable switches. Experiments show that μFAB can keep VF predictable under various network conditions, even under highly dynamic workloads, *e.g.*, incast. In compute scenario, under the dynamic background MongoDB traffic, μFAB supports 2.5× higher QPS and cuts 20× tail latency for Memcached compared with alternatives. In the storage scenario, μFAB can efficiently reconcile different tasks and decrease end-to-end storing time 4.8× on average and more than 21× at the tail. Meanwhile, μFAB can support tens of thousands of VM-pairs with <20% extra hardware resources and <1.28% probing bandwidth overhead, benefited by a scalable probing scheme. NS3 simulations verify that μFAB can keep sub-millisecond convergence under real workload in large-scale topology (512 servers).

Claim: This work does not raise any ethical issues.

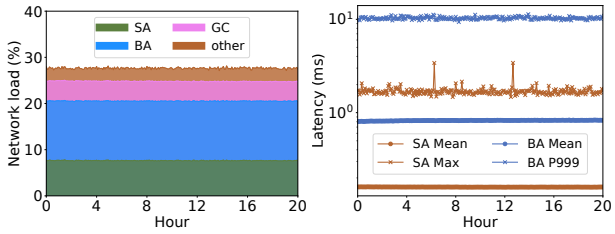
2 MOTIVATION

2.1 Practical challenges for predictable VFs

In production data centers, best-effort mechanisms are widely deployed to manage network resources. Specifically, (1) to suppress interference among traffic from different tenants, data center operators deliberately keep the overall network utilization low; (2) ECMP is used as the de-facto in-network load balance for splitting traffic on equivalent paths. However, without efficiently monitoring and reacting to complex in-network situations, two major obstacles make them perform poorly in practice.

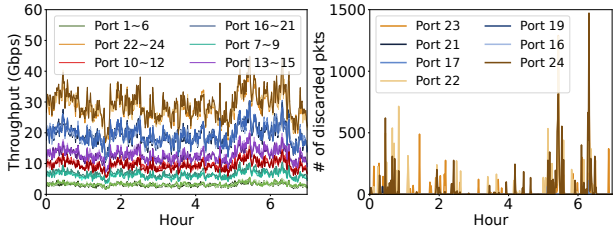
1. Traffic is bursty at short timescales. Traffic dynamic in DCNs shared by uncooperative tenants is changing more and more dramatically, which requires ensuring VF predictability for tenants even at short timescales. Despite the cloud provider typically overprovisioning the bandwidth capacity, it still cannot efficiently isolate network resources among tenants to accommodate the applications' increasingly sensitive demands at short timescales. Burst traffic from different tenants may not break the bandwidth restriction in the long term, but the corresponding accidental resource competitions lead to a large number of tail latency cases. We observe this phenomenon in both production compute and storage scenarios.

In the Elastic Compute Service (ECS) scenario, Figure 1 shows round-trip time (RTT) of a tenant's traffic from a cloud provider over one month, and hourly-averaged network utilization of all tenants'



(a) Network load (b) Task completion time

Figure 2: Bursty traffic interference in EBS scenario.



(a) Port throughput (b) Port discard

Figure 3: Load imbalance among equivalent upstream links of an Aggregation switch in production data center.

traffic in that cluster. While hourly-averaged network utilization is below 10%, the tenant observes a periodic bandwidth decline and up to 50× latency inflation in 99.9th percentile than the median, due to the traffic interference from another tenant (not shown) running routine data analytics.

In the Elastic Block Storage (EBS) scenario, several independent tasks work together to provide the complete storage service. Figure 2a shows the average network load of a server, which consists of (1) SA: requests/responses of Storage Agents towards block servers, (2) BA: three data replicas sent from Block Agents to chunk servers, (3) GC: periodically executed Garbage Collection service by merging data modification and comprises stored data. Each task in production is treated as from different "tenants" since they need individual network resources to satisfy performance consistently. The overall network utilization keeps steady at 27%, and each component shares a fixed part of bandwidth at the second granularity. However, Figure 2b shows that the tail task completion time is 10× larger than the average. Through comprehensive analysis, the root cause is that the great burst of traffic in the millisecond granularity exceeds the software processing capability and triggers the PCIe back-pressure, which finally leads to temporary queuing in the NIC.

2. It is nontrivial to aggregate multiple parallel paths as an ideal pipe. Previous work focusing on allocating the end-to-end network resource often assumes that an independent load balance mechanism can equally split traffic on all equivalent paths. However, load imbalance is quite common in production data centers. Figure 3a 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 ~ 24 bears 10× larger load

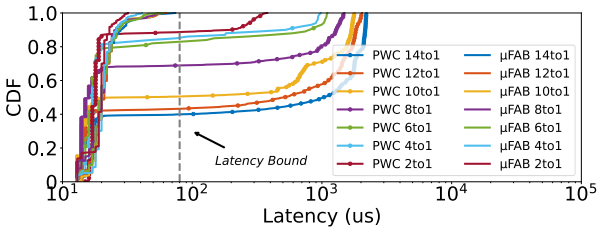


Figure 4: RTT under various incast degrees. µFAB can bound tail latency as the incast degree increases, while the tail latency of PicNIC′+WCC+Clove (PWC) increases. (Case-1)

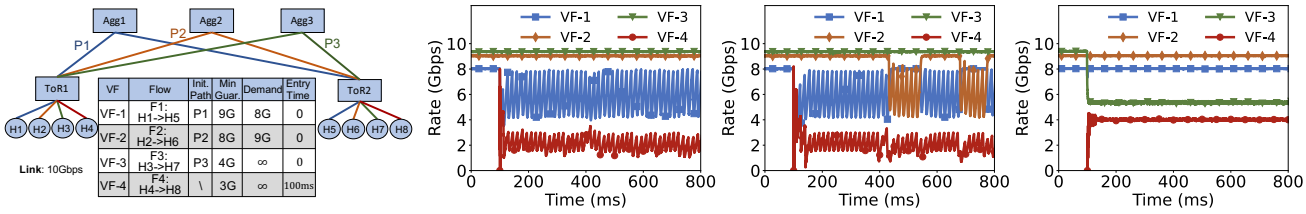
than link 1 ~ 6. The root cause is that both ToR switches and Aggregation switches use the same type of switch chip, which causes hash polarization [63]. Even worse, the deployed switching chip has few candidates for hash algorithms, and no combination of different layers can avoid this problem altogether. Besides hash polarization, many other reasons (*e.g.*, hash collision of elephant flows, specific routing configuration, or forwarding priority) could also lead to load imbalance. Hence, even if we assign correct weights for traffic from different VFs, their bandwidth guarantees could still not be satisfied owning to be allocated to congested paths.

2.2 State-of-the-art solutions cannot help out

While there is a gap between production-deployed and state-of-the-art solutions, we argue that they all use heuristic evolution based on limited network information. This root cause determines that prior solutions (and their combinations) cannot converge accurately and efficiently to the ideal results[3]. For example, PicNIC [37], the state-of-the-art scheme designed for predictable virtualized NIC, provides guaranteed performance at edges but cannot address fabric congestion. Seawall [51], a representative of runtime network-wide bandwidth allocation schemes, uses weighted congestion control (WCC) algorithms to share network bandwidth proportionally to the per-source weight but converges slowly (tens of milliseconds). WCC is widely deployed in typical bandwidth allocation schemes [25, 29, 45, 51], making them all fail to react to sub-millisecond-level traffic bursts. Clove [31] selects a path for flowlets based on explicit path utilization for load balancing. However, it cannot provide bandwidth guarantee for tenants due to the lack of tenant-level guarantee information. These drawbacks can significantly impact the predictability of end-to-end performance. Next, we use experiments to illustrate this point.

Experiment settings : Our testbed has the link capacity of 10Gbps, with the maximum base RTT (*baseRTT*) of 24μs. Like existing work [29, 40, 45], we set the target bandwidth utilization as 95%. Since we mainly focus on network resources, we only compare 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 [29]. We choose Swift [36], a delay-based CC recently proposed for DCN, as the basis of WCC, due to its excellent low latency. We use Clove [31] as the load balance mechanism to split traffic at the flowlet granularity according to the path utilization.

[3]Comparisons between µFAB and related work can be found in Appendix A.



(a) Flow bandwidth

(b) PicNIC′+WCC+Clove (200μs)

(c) PicNIC′+WCC+Clove (36μs)

(d) μFAB (Close to ideal)

Figure 5: Path migrations for utilization-oriented load balance may endanger minimum bandwidth guarantees. (Case-2)

Case-1: Greedy rate evolution may break the latency bound. We use an "incast" scenario to show that the combination of current solutions cannot guarantee bounded latency. N flows belonging to different VFs have the same destination host, and their bandwidth guarantees are all $500Mbps$. They start to transmit traffic at the same time. N increase from 2 to 14, and Figure 4 shows the distribution of RTT under different scenarios. The tail latency is positively associated with the incast degree. The root cause is that existing congestion control heuristically evolves flow 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: Utilization-oriented load balance may break bandwidth guarantee. As shown in Figure 5a, $F1, F2, and F3$ are initially allocated to different paths and achieve the desired bandwidth. At this point, the subscription of Path $P1$, $P2$, and $P3$ 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 demand ($P1$) or work conservation ($P2$ and $P3$). At $100ms$, F4 enters into the network. Since path $P1$ has the lowest utilization, $F4$ is assigned to $P1$. However, it causes bandwidth dissatisfaction of $F1$. With the recommended flowlet gap in Clove (200μs), $F1$ and $F4$ remain in $P1$, with bandwidth guarantee persistently unsatisfied. This result shows that the existing load balance mechanism cannot perceive the breaking of the bandwidth guarantee.

Furthermore, to force per-flowlet load balance functional in this case, we decrease the flowlet gap to 36μs ($1.5\times baseRTT$). So a small queue buildup would trigger path migration. In this case, as shown in Figure 5c, $F4$ never gets acquired 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 guaranteed bandwidth. Then $F4$ will try to move back to $P1$ again, since $P1$ has the lowest utilization now. $F4$ migrates paths under 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 end-to-end link utilization cannot reflect the essential bandwidth contention, *i.e.*, total minimum bandwidth subscription on a link. Therefore, bandwidth isolation in multi-path architectures is not orthogonal to load balancing, which might break assumptions by existing solutions [12, 45].

2.3 Solution: the edge-core fusion

In the traditional network architecture, typically, the network core (switches) works independently with the edge (end hosts), 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, abundant in-network information, previously inaccessible, can now be conveniently calculated, stored, and transmitted. Such information allows the edge to make timely decisions on data transmissions without going through the time-consuming and inaccurate heuristics. Some work leverages in-network information to improve network performance [31, 40], but none of them provides predictable VF for tenants.

This paper's core mission is to explore how to fundamentally improve the VF predictability via collaboration between an informative core and an active edge. This direction looks promising – for instance, Figure 4 and Figure 5d show that μFAB's behaviors are close to ideal in terms of both steady-state and convergence speed.

3 DESIGN

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

3.1 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 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 [12, 44, 49], we choose the idea of Guarantee Partitioning (GP) proposed by ElasticSwitch [45], which dynamically assigns the VM-to-VM bandwidth guarantees based on the VF's hose model and online traffic patterns. Hence, we focus on the VM-to-VM guarantees given by ElasticSwitch in this paper.

Design goals. μFAB has the following design goals:
(i) Minimum bandwidth guarantee. A VF quickly provisions the minimum bandwidth predefined by tenants for each vNIC that has sufficient traffic demand;
(ii) Work conservation. A VF allows each vNIC to swiftly go beyond the minimum bandwidth to fully utilize network resources if other vNICs do not have sufficient traffic demands;

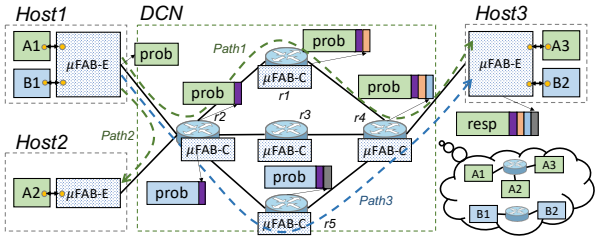


Figure 6: The overview of μFAB's system architecture.

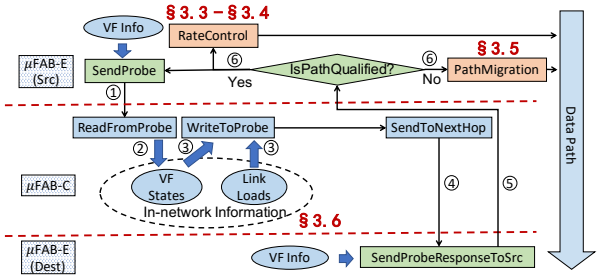


Figure 7: The overall workflow of μFAB.

(iii) Bounded tail latency. A VF bounds the end-to-end network latency between vNICs under bursty traffic demands;

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

Assumptions: *(i)* μFAB operates on the DCN which is constructed by the commodity programmable switches, *e.g.*, Barefoot Tofino and Broadcom Trident-4; *(ii)* μFAB assumes that the entire DCN topology is known as priori knowledge, thus μFAB-E knows all path candidates; *(iii)* Similar to prior work [25, 28, 45], μFAB assumes that VMs have been placed by other virtual cluster allocation algorithms, *e.g.*, Oktopus [12], so there are *theoretically* feasible solutions to satisfy the minimum bandwidth even in the worst case.

Non-assumptions: *(i)* μFAB does not assume a congestion-free network core and DCN topology can be single-path or multi-path, with or without over-subscription; *(ii)* μFAB does not assume a perfect load-balancing — the load balancing can have hot spots due to reasons like hash polarization or hash collision of elephant flows; *(iii)* μFAB does not assume any traffic patterns from the tenants. There can be unpredictable on-off traffic bursts, large scale incasts, or long persistent flows with occasional micro-bursts. *(iv)* μFAB does not assume a switch to have a large number of priority queues. Since queues are scarce resources and used for different purposes, μFAB only needs a single queue.

3.2 System overview

Critical telemetry data. Through our deliberate analysis and design, the following telemetry data of each link l in network is necessary and sufficient to ensure VF predictability.

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

(ii) Queue size. It reports the current queue status in switches. Source edge can timely reduce sending rate to control queuing latency when observing a queue is building up.

(iii) TX rate. It reflects the output rate of the switch port. Combined with queue size, source edge can perceive the gap between actual link load and link capacity, allowing to accurately adjust rate to quickly converge to the target utilization.

(iv) Total bandwidth subscription. It is the sum of the minimum bandwidth guarantees of all active VFs passing through the link, guiding source edge to determine whether a path can satisfy the minimum bandwidth guarantee in the worst case.

(v) 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 6 shows that μFAB installs edge agents (μFAB-E) and core agents (μFAB-C) into the DCN 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 $Host_3$.

At edge, μFAB-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 μFAB-E inserts local VF information, *i.e.*, minimum bandwidth and sending window, into the probe. Along the forwarding path, the μFAB-C put the aggregated VF information, *i.e.*, total bandwidth subscription and total sending window, and network information, *i.e.*, link capacity, queue size and TX rate into the probe via INT. Destination μFAB-E returns all information piggybacked in the probe with a response, together with its local minimum bandwidth. The source μFAB-E compares the minimum bandwidth of the destination with its local one to determine the minimum bandwidth guarantee for the VM-pair.

Workflow. As shown in Figure 7, each μFAB-E sends probes along each active underlay path (Step ①). After the probe reaches a μFAB-C, the μFAB-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). Next, the probe is forwarded along the path until reaching the destination (Step ④). When the response sent by destination μFAB-E gets back (Step ⑤), source μFAB-E will decide whether to continuously use the path with a rate adjustment based on the information provided in the response or start to migrate to other path if the current path is not qualified anymore (Step ⑥). Table 2 in Appendix B lists the notations used in this paper.

3.3 μFAB-E: bandwidth allocation

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

Guaranteeing minimum bandwidth. We define $\phi_{a\to b}$ as the bandwidth token allocated by ElasticSwitch for VM-pair $a \to b$ [4] and B_u as the minimum bandwidth a unit token can give to a VM-pair. Hence, the minimum bandwidth $a \to b$ can get is $B_{a\to b} = B_u \times \phi_{a\to b}$. The sender VM a needs to choose a path and control its sending rate towards the receiver VM b to satisfy $B_{a\to b}$.

[4] § 6 discusses how to distribute VM's tokens among VM-pairs.

The strategy μFAB 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 [5], 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:

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

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

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 ($r_{a\to b} \geq B_u \times \phi_{a\to b}$). Otherwise, none of the VM pairs can achieve its minimum bandwidth guarantee. Therefore, when the sender a finds $C_l \geq \Phi_l \times B_u$, it can safely send traffic with $r_{a\to b}$ on link l because others will be satisfied with their minimum bandwidth guarantee too; If the sender a predicts that $C_l < (\Phi_l + \phi_{a\to b}) \times B_u$ after VM pair $a \to b$ joins in the link l, it will not continue this migration, 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 $r_{a\to b}$, because some senders might not always have data to send, *i.e.*, insufficient demands. For work conservation, senders should use $r_{a\to b}$ as a lower bound, and it can go beyond that. We define $R_{a\to b}$ as the upper bound bandwidth that VM-pair $a \to b$ can have, which varies according to the actual load of the core. 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:

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

where $R_l = \sum_{\forall a\to b\in P_l} R^l_{a\to b}$ is also reported by μFAB-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 and keep the proportional ($\frac{\phi_{a\to b}}{\Phi_l}$) sharing among VM-pairs. Similarly, we have $R_{a\to b} = \min_{l\in p_{a\to b}}\{R^l_{a\to b}\}$, because we cannot utilize more bandwidth than the bottleneck link.

3.4 μFAB-E: 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 [27, 40]. Then, we modify Eqn (2) to the following:

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

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

Bounding the worst-case latency : While $w_{a\to b}$ allows immediate use of network capacity if the window permits, it cannot handle transient congestion during synchronized bursts. Especially, when the link keeps being under-utilized, it is quite possible that $w_{a\to b} = C_l \times T_{a\to b}$ 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 guarantee quickly and a little slower to converge to work conservation. This is because tenants pay for its guarantee, 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 $w'_{a\to b} = \phi_{a\to b} \times B_u \times T_{a\to b}$ as bootstrap sending window. It then performs additive increasing by $w'_{a\to b} \leftarrow w'_{a\to b} + \frac{\phi_{a\to b}}{\Phi_l} \times C_l \times T_{a\to b}$ per RTT until $w'_{a\to b}$ is larger than the $w_{a\to b}$ from Eqn (3) and then start to use $w_{a\to b}$.

Scenario-2 : For an existing VM-pair on a path but with actual sending rate lower than $r_{a\to b}$, it first uses $w'_{a\to b} = r_{a\to b} \times T_{a\to b}$ 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 than *Scenario-1*.

For a link l, in the worst case, all VM pairs start simultaneously (in *Scenario-2*) and increase one l's BDP per RTT. Our theoretical analysis (in Appendix C) suggests senders take 2 RTTs to learn the load created by the initial burst from the core and start to reduce its sending rate, the maximum inflight bytes is bounded by:

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

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

3.5 μFAB-E: path migration

While μFAB-C can detect performance degradation and migrate to other paths swiftly (we demonstrate it later), μFAB-E needs to avoid the side-effect caused by path migration.

Triggers of path migration : In μFAB there are two reasons for path migrations: (i) Satisfying minimum performance requirements

[5] $C_l = \eta C'_l$. C'_l is l's physical bandwidth capacity and we pick $\eta = 0.95$ to absorb transient bursts.

when the current path becomes incapable; (ii) 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 different emergency. (i) should be done quickly with caution – 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 (ii) should be performed less frequently – a VM-pair should observe a persistently better path for a long duration (30 seconds in our implementation) before migrating.

Path selection : While μFAB-E can perceive all the underlying paths between the source and the destination, it randomly chooses a few of them as candidate paths for the VM-pair. When a VM-pair initially joins in or begins a path migration, μFAB-E sends multiple probes to the candidate 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 (ii), 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, μFAB-E only allows one path migration in a randomly picked freeze window within $[1, N]$ RTTs on each host. Hence, a μFAB-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 [34] and our evaluation.

Avoiding reordering : Though not all tenants will require to prevent packet reordering during path migrations, μFAB still offers an option to do that. If this option is enabled, μFAB-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.

3.6 μFAB-C: informative core

Summarizing bandwidth demands : While tx_l, q_l, and C_l are straightforward to obtain by programmable switches [40], 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 activated and when it turns to inactive, which is almost impossible to 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 $a \to b$ takes its current $\phi_{a\to b}$ and $w^l_{a\to b}$, 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 $\phi_{a\to b}$ and $w^l_{a\to b}$ have been seen. If not, it will add $\phi_{a\to b}$ and $w^l_{a\to b}$ 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. μFAB-C also handles silent quits, and consumes limited hardware resources (§4.2).

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 $r_{a\to b}$ a little bit, it does not influence the proportional sharing and work conservation. A larger $r_{a\to b}$ 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 minimum bandwidth. 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, μFAB-C is open to leverage other advanced streaming algorithms, such as timing Bloom filter [61], for better efficiency.

4 IMPLEMENTATION

We fully implement μFAB with smart NICs and programmable switches. We have two versions for the active edge: one is on ARM-based SoC smart NICs, and the other is on FPGA-based smart NICs. The SoC version can fully support any transport stack and application software transparently, while the FPGA version is used to evaluate μFAB's complexity, efficiency, and performance running in hardware[6]. The SoC smart NIC consists of eight 3.0 GHz embedded processors, 16 GB DRAM, and PCIe Gen3×8 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, Xilinx Alveo U200 card, provides line-rate packet processing for a 100G port. It has a 64 GB onboard DRAM and a PCIe Gen3×16 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 μFAB-C in a Barefoot Tofino programmable Ethernet switch with 32x100G ports. It has a CPU of four 2.2 GHz cores, 16 GB DRAM, and a PCIe Gen2×4 bus. The implementation has ~3400 lines of P4 code and 3600 lines of python configurations in the control plane.

4.1 μFAB-E at smart NICs

The following descriptions focus on the FPGA-based implementation. There is a slight difference that μFAB-E in FPGA supports Verbs interface [1] with DMA, while we use DPDK in SoC to transparently support socket-based transports.

Figure 8 shows the μFAB-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 bandwidth token and sending window. *Path Monitor*

[6]The FPGA version has not transparently supported socket-based applications yet due to project scheduling.

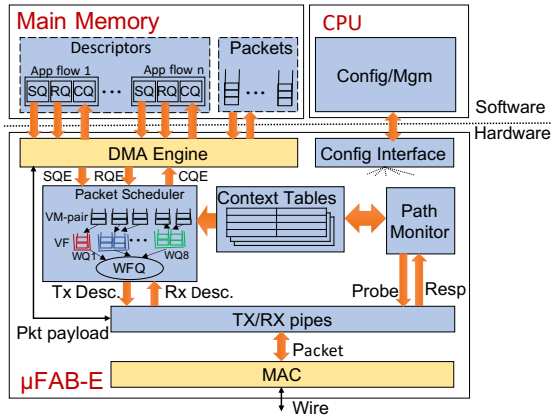


Figure 8: μFAB-E on FPGA-based SmartNIC.

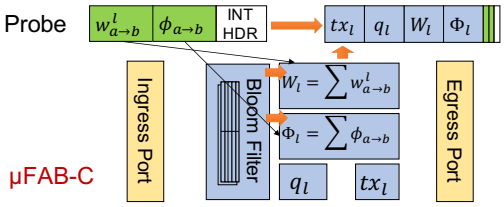


Figure 9: μFAB-C on P4 programmable switch.

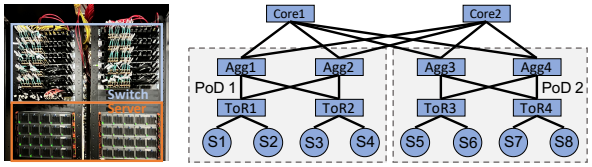


Figure 10: Testbed topology.

maintains the reachable paths discovered by the existing route discovery component, *e.g.*, source routing controller, and continuously monitors the path quality and performs path migration.

μFAB-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 source routing. The software then follows the standard Verbs APIs to exchange packets and descriptors with μFAB-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. Consecutive network predictability violations or drops of probes will trigger path migration. Since the inflight bytes is no more than three times of BDP, the latency is bounded by 4 $baseRTT$s, which is the sum of $3BDP/C_l$ and $baseRTT$. Hence, we detect probe loss by timeout beyond 8 $baseRTT$s.

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.

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, μFAB sends probes only when a VM-pair has immediate 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.*, one 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. In practice, this optimization still retains a good track of information in the core.

Resource usage : Our prototype of μFAB-E based on FPGA supports 8K VM-pairs and 1K tenants, with up to 10% extra hardware resources, *e.g.*, Registers, LUTs and Block RAM (more details can be found in Table 3 in Appendix H).

4.2 μFAB-C at programmable switches

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

Telemetry : As described in § 3.2, μFAB-C reads 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 diameter of 5 hops, the total size of the telemetry data is less than 100 bytes, which has low overhead. Appendix G shows the specific packet format.

Information summary : To recognize active VM-pairs for computing Φ_l and W_l, μFAB-C adopts a Bloom filter with two memory banks running in parallel. With a 2-way hashing Bloom filter of 20 KB, μFAB-C supports a moderate of 20K distinct VM-pairs with less than 5% false positives, while those false positives have limited impacts as presented in §3.6.

Handling silently inactive VM-pairs : μFAB 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, μFAB-C periodically (10 *sec* 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.

Resource usage : To support 20K distinct VM-pairs with tens of thousands of flows, most types of hardware resources, *e.g.*, SRAM,

PHV, Hash Bits, consumed by μFAB-C are less than 20%. Moreover, with the increase in the scale of VM-pairs, the hardware resource consumption only increases slightly, making μFAB scalable based on commodity programmable switches. More details can be found in Table 4 in Appendix H.

5 EVALUATION

We use testbed experiments with our prototypes and large-scale NS3 simulations [52] to evaluate μFAB performance.

5.1 Evaluation setup

Environment : As shown in Figure 10, 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. We use virtio vNIC, while μFAB supports multiple types of vNIC. 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 [6] with 100Gbps links. We use 16 and 32 Core switches to set an oversubscription ratio of 1:2 and 1:1, respectively. All links in simulations are 100Gbps with 1 μs propagation delay, and the target bandwidth utilization is set as 95%.

Alternatives : We compare μFAB 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.

5.2 Microbenchmarks

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

Bandwidth guarantee with work conservation : This experiment shows how μFAB and the alternatives handle intensive minimum bandwidth demands. For this, a permutation traffic pattern with different bandwidth guarantees is generated. There are three classes of VFs: minimum bandwidth guarantee 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 = 8Gpbs < 10Gbps). We randomly insert a VF every 20 *ms* to evaluate the convergence and bandwidth guarantee properties.

Figure 11a shows that μFAB achieves fast convergence when a new VF joins in. It persistently guarantees the minimum bandwidth with work conservation for all VFs via distributed traffic admission and path migration. In contrast, PicNIC′+WCC+Clove (Figure 11b) 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 11c) guarantees bandwidth for more VFs, but causes serious queuing in the network (Figure 11e), because it uses the minimum bandwidth as a lower bound of sending rate, even if the network is congested.

Figure 11d shows the bandwidth dissatisfaction ratio, the amount of minimum bandwidth violation over the total traffic volume. 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. In contrast, μFAB efficiently adjusts VFs to proper paths and rapidly converges to steady bandwidth sharing. Hence, the dissatisfaction ratio is mostly close to zero even when VFs continuously join. Figure 11e shows that μFAB always keeps queue size low even with the continuous injection of new VFs, due to its excellent performance in terms of convergence speed.

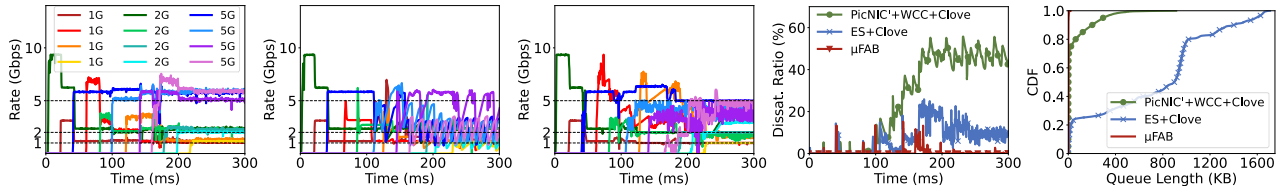
Bounded Latency : We extend the 14-to-1 incast experiment in Case-1 with more baselines. Figure 12a shows how different solutions react to incast. μFAB′ is the one without the bounded latency optimization. Both μFAB and μFAB′ quickly react to incast, and all VFs efficiently converge to the steady rate, while both PicNIC′+WCC+Clove and ElasticSwitch+Clove converge slowly with rate fluctuation. Figure 12b 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 μFAB′ decreases it by 11× with feedback from the informative core. Still, these schemes cannot bound tail latency. With the latency optimization, μFAB can restrain burst and control tail latency under the expected bound.

5.3 Application-level performance

We further evaluate application-level performance with transparent support of μFAB.

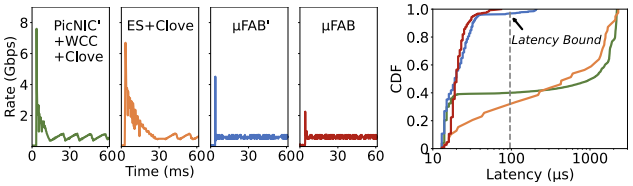
Multiple ECS tenants. We set up two tenants: one tenant creates a VF to run Memcached, a latency-sensitive application, while the other creates another VF to deploy MongoDB, which is bandwidth-hungry. Memcached places 24 VMs as servers evenly over S7-S8, and 12 VMs as clients evenly over S1-S4. MongoDB places 24 VMs as servers evenly over S5-S8, and 24 VMs as clients evenly into S1-S4. Each MongoDB client continuously fetches 500KB data from a random DB server. Each Memcached client periodically fetches data from random servers, where the data size follows an empirical distribution of key-value workload [10] with a mean size of 2KB. Note that the tenants compete for bandwidth in both edge and network core. Similar to [37], we focus on Memcached's performance due to its vulnerability to bandwidth contention. μFAB transparently supports the tenants with predictable VF on bandwidth and latency. Figure 13 shows the QPS (Query per Second) and QCT (Query Completion Time) of Memcached requests with different workloads. μFAB achieves QPS and QCT similar to the ideal case, *i.e.*, without MongoDB traffic, because μFAB can find proper paths for different VFs and react to fabric congestion rapidly. In contrast, the alternatives cannot isolate the traffic of different tenants well, which leads to 2.5× lower QPS and 20× higher tail QCT.

Multiple tasks in EBS. We deploy EBS applications in our testbed: S1-S4 each have a VM as a Storage Agent, while S5-S8 each have three VMs: Block Agent, Chunk Server and Garbage Collection Agent. SA sends a 64KB message to a random BA every 320μs, and BA replicates the received data to three CS after receiving the whole message. At the same time, GC reads data from a random CS every 1ms, and then writes the compressed data back. Considering the linerate in our testbed is 10× lower than production data center, the requirements for the latency of an EBS I/O operation is



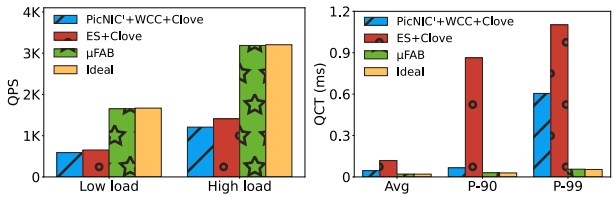
(a) Rate (µFAB) (b) Rate (PicNIC′+WCC+Clove) (c) Rate (ES+Clove) (d) Bandwidth dissatisfaction (e) Queue length (CDF)

Figure 11: Bandwidth evolution under high load. Bandwidth dissatisfaction indicates total minimal bandwidth violation.



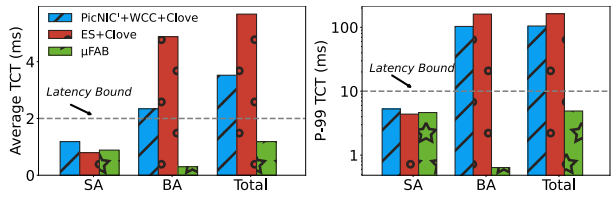
(a) Rate evolution (b) Network RTT

Figure 12: Incast performance (bounded latency).



(a) QPS (b) QCT under high load

Figure 13: Performance of Memcached.



(a) Average task completion time (b) Tail task completion time

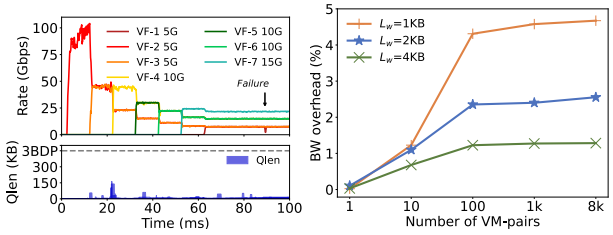
Figure 14: Performance of EBS. Latency bound is converted under 10Gbps.

converted to 2*ms* on average and 10*ms* at tails. Figure 14 shows the TCT (task completion time) of different tasks when the minimum bandwidth guarantees for SA, BA, and GC are 2Gbps, 6Gbps, and 1Gbps, respectively. Benefit from accurate path selection and rate control, µFAB can complete I/O operations within the required latency bound, and achieving 21× and 33× shorter tail total TCT than PicNIC′+WCC+Clove and ElasticSwitch+Clove, respectively.

5.4 Hardware performance and scalability

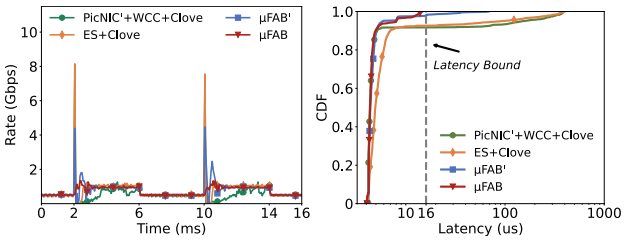
We use the FPGA-based prototype to show that µFAB is capable of supporting line-rate of 100GE with small overhead.

Support 100GE network at line-rate : Figure 15a shows that µFAB scales naturally to support 100GE and ensures predictability upon



(a) Fabric predictability (b) Bandwidth overhead in probing

Figure 15: Fabric predictability and overhead in 100GE.



(a) Rate evolution (b) RTT (CDF)

Figure 16: Performance with 90-to-1 dynamic workload.

traffic dynamics and failures. 7 VFs towards S8 but with different minimum bandwidth guarantees continuously enter the network every 10 *ms*. µFAB 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 µFAB swiftly migrates victim VFs to other paths. Despite the bursts and failures, µFAB continuously maintains a close-to-zero queue.

Bounded probing overhead : We use one VF to generate persistent traffic to saturate the outgoing link and measure the bandwidth overhead using probes. As shown in Figure 15b, with the increasing number of VM-pairs, bandwidth overhead gradually goes steady (as suggested in § 4.1). By setting L_w = 4KB, the upper bound of overhead is 1.28%. Therefore, µFAB can smoothly scale up to support more VM-pairs.

5.5 Convergence in large scales

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

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

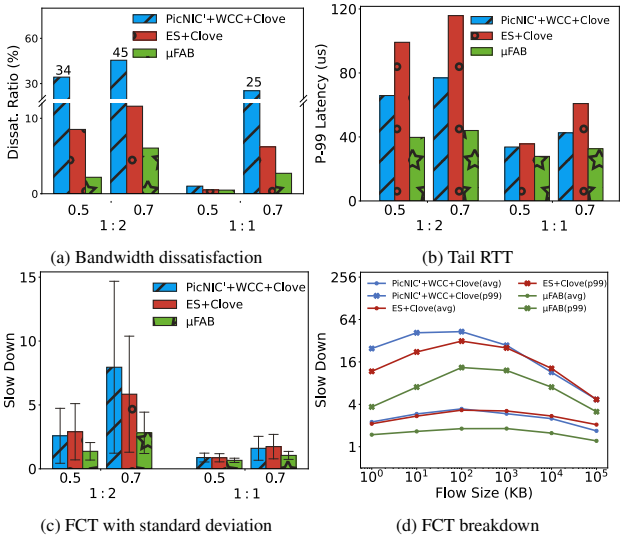
Performance under real workload: We generate tenant VFs with random minimum bandwidth guarantees. The number of VMs per tenant and the number of destinations each VM communicates at runtime are synthesized from empirical production data centers [14]. The distribution of flow size is consistent with empirical workload [7], 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 [28]. Figure 17a shows that the bandwidth dissatisfaction of μFAB 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 guarantee, it causes higher tail latency (Figure 17b). In contrast, μFAB efficiently guarantees the minimum bandwidth while controlling the network congestion, because it can adjust sending rate and select path accurately and rapidly. Hence, μFAB achieves much better FCT slowdown[7], as shown in Figure 17c. Further, Figure 17d shows the FCT breakdown for flows with different flow sizes under 1:1 oversubscription network and load of 0.7. We omit other scenarios since their FCT breakdown results are similar.

5.6 Sensitivity analysis

Path migration freeze window: Figure 18a and Figure 18b shows insights on how μFAB'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%, μFAB's slow migration policy shows its advantage especially in decreasing path migration frequency. We select 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 18c shows that with periodic probing, *e.g.*, probe packets are sent every n RTTs, μFAB 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.

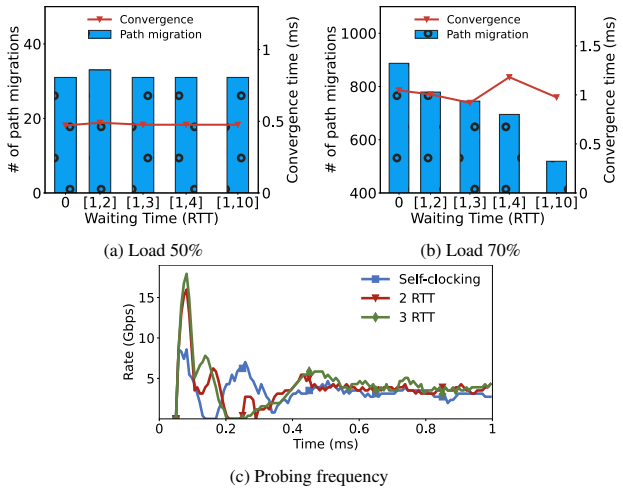
[7]FCT slowdown means the actual FCT normalized by the expected FCT, *i.e.*, dividing flow size by VM's minimum bandwidth guarantee in Hose model.



(a) Bandwidth dissatisfaction (b) Tail RTT

(c) FCT with standard deviation (d) FCT breakdown

Figure 17: Performance under real workload with 1:2, 1:1 oversubscription network and loads of 0.5 and 0.7.



(a) Load 50% (b) Load 70%

(c) Probing frequency

Figure 18: Convergence and stability.

6 DISCUSSION

Explicit bandwidth allocation : μFAB 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 [17]. 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.

The number of underlay paths : The traditional path management in DCN spreads the flows into numerous underlay paths, while μFAB only assigns a small number of underlay paths (even a single path in the majority of this paper) for a VM-pair. Obviously, μFAB's

way has smoother traffic and is more manageable. Also, μFAB is responsive to capacity issues because its underlay paths are dynamic to protect performance. In high bisection DCNs, μFAB can even use a single dynamic path for each VM-pair, while it indeed needs more paths in oversubscribed DCNs to utilize more bandwidth[8].

Bandwidth token assignment : μFAB is open to any bandwidth tokens assignment algorithms that dynamically assign VM's tokens to VM-pairs. For concreteness, μFAB adopts the idea of GP in ElasticSwitch [45]. μFAB partitions a VM's token among its remote peers, and the unused tokens contributed by bounded VM-pairs, which have insufficient traffic demand, are reassigned to unbounded ones. We use a per-VM-pair rate meter to estimate demand. Appendix E describes the algorithm in detail for completeness.

Impact of heterogeneous response delays : The self-clocking probing scheme may cause μFAB-Es to receive responses that are out of sync. The theoretical analysis in Appendix C.3 shows that μFAB-Es can still converge even though they react to different views of network information. In a 128-to-1 incast simulation where different μFAB-Es receive responses asynchronously, μFAB can still converge quickly[9]. The impact of heterogeneous response delays in more dynamic and large-scale production data centers deserves further study and validation.

Cooperation with other control loops in the network : Similar to prior VF solutions [37, 45], μFAB-E works in physical machines and can coexist with any congestion control scheme used by guest VMs, while it is not necessary for the guest VMs to use congestion control. μFAB works with any per-flow load balancing solution, *e.g.*, ECMP, which is widely deployed in today's DCNs. We leave the deep analysis of cooperating with other solutions, *e.g.*, per-packet [15], per-flowlet [7, 55] or per-flowcell [22] load balancing, as our future work.

Comparison with centralized solutions : Centralized solutions, *e.g.*, Fastpass [43], schedule packets based on a global view of a central arbiter. While the global information can theoretically help improve traffic admission and path selection decisions, interaction with a central arbiter can worsen the latency for short flows. By contrast, with μFAB-C aggregating information from different VFs globally, μFAB-E controls traffic admission and path selection in a distributed manner to improve the latency for short flows.

Partial deployment of programmable hardware : Partial deployment of μFAB-C may lead to incomplete in-network information and degrade the overall performance guarantee. On the other hand, if programmable NICs are unavailable, μFAB-E can also interact normally with μFAB-C as a component in the hypervisors.

7 RELATED WORK

Predictable virtual fabric has been an enduring research topic. μFAB is the first that leverages the informative data plane to ensure the end-to-end VF predictability in bandwidth guarantee, work conservation and bounded latency simultaneously.

Bandwidth : Static bandwidth reservation [12, 21, 28, 39] provides strong guarantees but with low resource utilization. FairCloud (PS-P) [44] and NetShare [38] require per VM/tenant queues for bandwidth allocation, which is infeasible in commodity switches; Seawall [51] uses TCP-like algorithms to dynamically allocate bandwidth. FairCloud (PS-L/N) [44] and ElasticSwitch [45] 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 [44]. Edge-based solutions, *e.g.*, PicNIC [37], EyeQ [29] and GateKeeper [49], 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 always held in practice.

Latency : Many solutions achieve low latency using queue reduction [8, 36, 40], traffic prioritization [9, 42], deadline-aware scheduling [23, 53, 59], in dedicated networks. However, those works do not directly provide bounded latency in multi-tenant DCNs. Silo [28] uses network calculus, and Chameleon [54] further exploits path diversity to bounded packet delay, QJUMP [20] uses priority queuing and rate-limiting. All above approaches are static allocation and lack of work conservation. Aquila [19] leverages a receiver-driven solicitation scheme and a custom cell-switched substrate to achieve low latency. PicNIC [37] and Justitia [62] consider ingress/egress engine processing capacity isolation at the end host. Based on ElasticSwitch, Trinity [26] leverages priority queues to prioritize the traffic of bandwidth guarantee over that of work conservation, thereby achieving low latency for short flows. By contrast, μFAB limits the burst size to achieve both bounded latency and work conservation simultaneously, with a single queue.

Informative core : Existing works leverage the network information from programmable switches for many aspects, such as HPCC [40] for congestion control, Clove [31] for load balancing, NetSeer [64] for monitoring, *etc.* By contrast, μFAB is the first that defines the critical information for providing predictable fabric service and builds a dynamic-path framework with active edge and informative core fusion.

8 CONCLUSION

We believe the newly available programmable data plane is the key to solving the exceptional challenges of providing predictable virtual fabric in multi-tenant DCNs – μFAB is such an example. μFAB constructs a predictable virtual 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 predictable tenant-level performance (*e.g.*, guaranteed bandwidth and bounded latency) and high utilization at sub-millisecond timescales. We demonstrate that μFAB can be efficiently implemented with commodity smart NICs and programmable switches.

ACKNOWLEDGMENT

We thank our shepherd Dr. Lalith Suresh and SIGCOMM reviewers for their constructive feedback. Prof. Dan Li is the corresponding author. This work was supported by the National Key Research and Development Program of China under Grant 2019YFB1802600, the National Natural Science Foundation of China under Grant U21B2022, Alibaba Innovative Research (AIR) Program, and Tsinghua University-China Mobile Communications Group Co.,Ltd. Joint Institute.

[8] Appendix F shows how μFAB scales to multiple underlay paths.
[9] More details about the simulation settings and results can be found in Appendix D.

REFERENCES

[1] 2015. InfiniBand Architecture Specification Volume 1 Release 1.3. *InfiniBand Trade Association (IBTA)* (2015).

[2] 2020. Alibaba Cloud EBS performance. https://www.alibabacloud.com/help/doc-detail/25382.htm. (2020).

[3] 2020. Amazon EBS features. https://aws.amazon.com/ebs/features/. (2020).

[4] 2020. Google Cloud Block Storage Performance. https://cloud.google.com/compute/docs/disks/performance. (2020).

[5] 2020. Microsoft Azure Disk Storage. https://azure.microsoft.com/en-us/services/storage/disks/. (2020).

[6] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM CCR* (2008).

[7] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, et al. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM*.

[8] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *ACM SIGCOMM*.

[9] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM CCR* (2013).

[10] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS*.

[11] Hamsa Balakrishnan, Nandita Dukkipati, Nick McKeown, and Claire J Tomlin. 2007. Stability analysis of explicit congestion control protocols. *IEEE Communications Letters* (2007).

[12] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2011. Towards predictable datacenter networks. In *ACM SIGCOMM*.

[13] Hitesh Ballani, Keon Jang, Thomas Karagiannis, Changhoon Kim, Dinan Gunawardena, and Greg O'Shea. 2013. Chatty tenants and the cloud network sharing problem. In *USENIX NSDI*.

[14] Peter Bodík, Ishai Menache, Mosharaf Chowdhury, Pradeepkumar Mani, David A Maltz, and Ion Stoica. 2012. Surviving failures in bandwidth-constrained datacenters. In *ACM SIGCOMM*.

[15] Jiaxin Cao, Rui Xia, Pengkun Yang, and al et. 2013. Per-packet load-balanced, low-latency routing for clos-based data center networks. In *ACM CoNEXT*.

[16] Mosharaf Chowdhury, Zhenhua Liu, Ali Ghodsi, and Ion Stoica. 2016. HUG: Multi-Resource fairness for correlated and elastic demands. In *USENIX NSDI*.

[17] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown. 2005. Processor sharing flows in the internet. In *Springer IWQos*.

[18] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yaohui Wu, Shaozong Liu, Lei Yan, et al. 2021. When Cloud Storage Meets {RDMA}. In *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*. 519–533.

[19] Dan Gibson, Hema Hariharan, Eric Lance, Moray McLaren, Behnam Montazeri, Arjun Singh, Stephen Wang, Hassan M. G. Wassel, Zhehua Wu, Sunghwan Yoo, Raghuraman Balasubramanian, Prashant Chandra, Michael Cutforth, Peter Cuy, David Decotigny, Rakesh Gautam, Alex Iriza, Milo M. K. Martin, Rick Roy, Zuowei Shen, Ming Tan, Ye Tang, Monica Wong-Chan, Joe Zbiciak, and Amin Vahdat. 2022. Aquila: A unified, low-latency fabric for datacenter networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1249–1266. https://www.usenix.org/conference/nsdi22/presentation/gibson

[20] Matthew P Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert NM Watson, Andrew W Moore, Steven Hand, and Jon Crowcroft. 2015. Queues Don't Matter When You Can JUMP Them!. In *USENIX NSDI*.

[21] Chuanxiong Guo, Guohan Lu, Helen J Wang, and al et. 2010. Secondnet: a data center network virtualization architecture with bandwidth guarantees. In *ACM CoNEXT*.

[22] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM CCR* (2015).

[23] Chi-Yao Hong, Matthew Caesar, and P Brighten Godfrey. 2012. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM CCR* (2012).

[24] Christian Hopps et al. 2000. *Analysis of an equal-cost multi-path algorithm*. Technical Report. RFC 2992, November.

[25] Shuihai Hu, Wei Bai, Kai Chen, Chen Tian, Ying Zhang, and Haitao Wu. 2018. Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud. *IEEE Transactions on Cloud Computing* (2018).

[26] Shuihai Hu, Wei Bai, Kai Chen, Chen Tian, Ying Zhang, and Haitao Wu. 2018. Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud. *IEEE Transactions on Cloud Computing* (2018).

[27] Van Jacobson. 1988. Congestion avoidance and control. *ACM SIGCOMM CCR* (1988).

[28] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. 2015. Silo: Predictable message latency in the cloud. In *ACM SIGCOMM*.

[29] Vimalkumar Jeyakumar, Mohammad Alizadeh, David Mazières, Balaji Prabhakar, Albert Greenberg, and Changhoon Kim. 2013. EyeQ: Practical network performance isolation at the edge. In *USENIX NSDI*.

[30] Dina Katabi, Mark Handley, and Charlie Rohrs. 2002. Congestion control for high bandwidth-delay product networks. In *ACM SIGCOMM*.

[31] Naga Katta, Aditi Ghag, Mukesh Hira, Isaac Keslassy, Aran Bergman, Changhoon Kim, and Jennifer Rexford. 2017. Clove: Congestion-aware load balancing at the virtual edge. In *ACM CoNEXT*.

[32] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. Hula: Scalable load balancing using programmable data planes. In *SOSR*.

[33] Frank Kelly, Gaurav Raina, and Thomas Voice. 2008. Stability and fairness of explicit congestion control with small buffers. *ACM SIGCOMM CCR* (2008).

[34] Frank Kelly and Thomas Voice. 2005. Stability of end-to-end algorithms for joint routing and rate control. *ACM SIGCOMM CCR* (2005).

[35] Frank Kelly and Elena Yudovina. 2014. *Stochastic networks*. Cambridge University Press.

[36] Gautam Kumar, Nandita Dukkipati, Keon Jang, and al et. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *ACM SIGCOMM*.

[37] Praveen Kumar, Nandita Dukkipati, Nathan Lewis, Yi Cui, Yaogong Wang, Chonggang Li, Valas Valancius, Jake Adriaens, Steve Gribble, Nate Foster, et al. 2019. PicNIC: predictable virtualized NIC. In *ACM SIGCOMM*.

[38] Vinh The Lam, Sivasankar Radhakrishnan, Rong Pan, Amin Vahdat, and George Varghese. 2012. Netshare and stochastic netshare: predictable bandwidth allocation for data centers. *ACM SIGCOMM CCR* (2012).

[39] Jeongkeun Lee, Myungjin Lee, Lucian Popa, Yoshio Turner, Sujata Banerjee, Puneet Sharma, and Bryan Stephenson. 2013. Cloudmirror: Application-aware bandwidth reservations in the cloud. In *USENIX HotCloud*.

[40] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *ACM SIGCOMM*.

[41] Jeonghoon Mo and Jean Walrand. 2000. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on networking* (2000).

[42] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A receiver-driven low-latency transport protocol using network priorities. In *ACM SIGCOMM*.

[43] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A centralized" zero-queue" datacenter network. In *ACM SIGCOMM*.

[44] Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. 2012. FairCloud: Sharing the network in cloud computing. In *ACM SIGCOMM*.

[45] Lucian Popa, Praveen Yalagandula, Sujata Banerjee, Jeffrey C Mogul, Yoshio Turner, and Jose Renato Santos. 2013. Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing. In *ACM SIGCOMM*.

[46] Bozidar Radunovic and Jean-Yves Le Boudec. 2004. Rate performance objectives of multihop wireless networks. *IEEE transactions on Mobile computing* (2004).

[47] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. *arXiv preprint arXiv:2201.05596* (2022).

[48] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[49] Henrique Rodrigues, Jose Renato Santos, Yoshio Turner, Paolo Soares, and Dorgival O Guedes. 2011. Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks. *WIOV* (2011).

[50] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. 2019. Scaling distributed machine learning with in-network aggregation. *arXiv preprint arXiv:1903.06701* (2019).

[51] Alan Shieh, Srikanth Kandula, Albert G Greenberg, Changhoon Kim, and Bikas Saha. 2011. Sharing the Data Center Network.. In *USENIX NSDI*.

[52] NS3 Simluator. 2021. NS3. https://www.nsnam.org/. (2021).

[53] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. 2012. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM CCR* (2012).

[54] Amaury Van Bemten, Nemanja Đerić, Amir Varasteh, Stefan Schmid, Carmen Mas-Machuca, Andreas Blenk, and Wolfgang Kellerer. 2020. Chameleon: predictable latency and high utilization with queue-aware and adaptive source routing. In *ACM CoNEXT*.

[55] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *USENIX NSDI*.

[56] Thomas Voice and Gaurav Raina. 2009. Stability analysis of a max-min fair Rate Control Protocol (RCP) in a small buffer regime. *IEEE transactions on automatic*

control (2009).
[57] Qiuping Wang, Jinhong Li, Patrick PC Lee, Tao Ouyang, Chao Shi, and Lilong Huang. 2022. Separating Data via Block Invalidation Time Inference for Write Amplification Reduction in Log-Structured Storage. In *USENIX FAST*.
[58] Shuai Wang, Dan Li, Jiansong Zhang, and Wei Lin. 2020. CEFS: compute-efficient flow scheduling for iterative synchronous applications. In *ACM CoNEXT*.
[59] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. 2011. Better never than late: Meeting deadlines in datacenter networks. *ACM SIGCOMM CCR* (2011).
[60] Di Xie, Ning Ding, Y Charlie Hu, and Ramana Kompella. 2012. The only constant is change: Incorporating time-varying network reservations in data centers. In *ACM SIGCOMM*. 199–210.
[61] Linfeng Zhang and Yong Guan. 2008. Detecting click fraud in pay-per-click streams of online advertising networks. In *IEEE ICDCS*.
[62] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. 2022. Justitia: Software Multi-Tenancy in Hardware Kernel-Bypass Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1307–1326. https://www.usenix.org/conference/nsdi22/presentation/zhang-yiwen
[63] Zhehui Zhang, Haiyang Zheng, Jiayao Hu, Xiangning Yu, Chenchen Qi, Xuemei Shi, and Guohui Wang. 2021. Hashing Linearity Enables Relative Path Control in Data Centers. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 855–862.
[64] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. 2020. Flow Event Telemetry on Programmable Data Plane. In *ACM SIGCOMM*.

APPENDIX

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

A RELATED WORK

Table 1 summarizes the previous works and μFAB. μFAB is the first that leverages the informative data plane to ensure the end-to-end predictability of VF in bandwidth guarantee, work conservation and bounded latency simultaneously.

B NOTATION

Table 2 lists the notations used in this paper, where the top part lists all the notations required by μFAB, while the ones in the bottom part help to illustrate the design of μFAB.

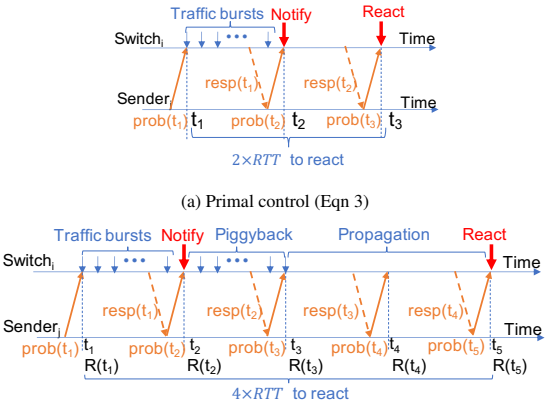
C THE ANALYSIS OF μFAB'S THEORETICAL PROPERTIES

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

C.1 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 $y = Ax \le C$ 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



(a) Primal control (Eqn 3)

(b) Dual control (Eqn 7)

Figure 19: Illustration of μFAB's convergence.

$$\text{maximize} \sum_j \frac{w_j}{1-\alpha}\left(\frac{x_j}{w_j}\right)^{1-\alpha} \quad \text{over } x \ge 0, Ax \le C. \tag{4}$$

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 $\alpha = 1$ [35, 41].

The solution to this optimization problem takes the form

$$x_j = w_j \left(\sum_i A_{ij} R_i^{-\alpha}\right)^{-1/\alpha} \tag{5}$$

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.

Note that as $\alpha \to \infty$ the expression (5) approaches

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

corresponding to weighted max-min fairness as used in §3. The case $\alpha = 1$ corresponds to weighted proportional fairness. If $w \equiv 1$ then as $\alpha \to 0$ the expression (5) approaches the rate allocation which maximizes the sum of rates over all paths [35].

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 [46].

C.2 Equilibrium rates and convergence

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

$$y(n) = Ax(n) \tag{6}$$

$$R_i(n+1) = R_i(n)\frac{C_i}{y_i(n)} \tag{7}$$

where the sending rates x are given by the expression (5). In discrete time, with exactly one RTT needed to update the sending rates, this corresponds to the update rule in §3.3. An equilibrium point

Systems	Strict Bandwidth Isolation			Low Latency	Multipath Awareness	Topology Requirement	Network Device Requirement
	Tenant-level Guarantee	Work Conservation	Convergence Speed				
QJUMP[20], Chameleon[54]	✗	✗	–*	✓	–‡	None	Priority queues
SecondNet [21], Oktopus [12], CloudMirror [39], Silo [28]	✓	✗	–†	✓	Static routing	None	None
Seawall [51], FairCloud [44] (PS-L/N)	✗	✓	10~200ms	✗	–‡	None	ECN
NetShare [38]	✗	✓	10~200ms	✗	–‡	None	Per-tenant queue
ElasticSwitch [45]	✓	✓	10~200ms	✗	Ideal assumption	None	ECN
Trinity [26]	✓	✓	10~200ms	Short flow	Ideal assumption	None	Priority queues, ECN
FairCloud [44] (PS-P)	✓	✓	10~200ms	✗	Ideal assumption	Tree topology	Per-tenant queue
Hadrian [13]	✓	✓	5~10ms	✗	Ideal assumption	Tree topology	Programmable
Proteus [60], EyeQ [29], HUG [16], GateKeeper [49]	✓	✓	5~10ms	✗	–‡	Congestion/loss-free fabric	None
PicNIC [37]	✓	Partial	5~10ms	Host side	–‡	Congestion/loss-free fabric	None
μFAB (Our work)	✓	✓	<1ms	✓	✓	None	Programmable

Table 1: Related systems. *The work cannot isolate network performance; †The work reserve bandwidth or use network calculus. ‡The work do not aim at providing tenant-level guarantee in the fabric.

Notation	Description	Source
B_u	Minimum bandwidth a unit token can give to a VM-pair	Prior knowledge
$T_{a\to b}$	baseRTT between a and b without queuing	Prior knowledge
Φ_l	Total bandwidth token of all active VM-pairs on link l	μFAB-C
C_l	Target bandwidth capacity for link l	μFAB-C
tx_l	Link l's actual TX rate	μFAB-C
q_l	Link l's real-time queue size	μFAB-C
W_l	Total sending window of all active VM-pairs traversing link l	μFAB-C
$\phi_{a\to b}$	Bandwidth token allocated for VM-pair $a\to b$	μFAB-E
$w_{a\to b}$	Sending window size of VM-pair $a\to b$ on path $p_{a\to b}$	μFAB-E
$w^l_{a\to b}$	Sending window size of VM-pair $a\to b$ on link l	Eqn (3)
$B_{a\to b}$	Minimum bandwidth guarantee for VM-pair $a\to b$	\
$p_{a\to b}$	Flow path of VM-pair $a\to b$	\
$r^l_{a\to b}$	Proportional bandwidth share of VM-pair $a\to b$ on link l	\
$r_{a\to b}$	Minimum proportional bandwidth share of VM-pair $a\to b$ along path $p_{a\to b}$	\
$R^l_{a\to b}$	Sending rate of VM-pair $a\to b$ on link l	\
$R_{a\to b}$	Sending rate of VM-pair $a\to b$ on path $p_{a\to b}$	\
R_l	Total sending rate of all active VM-pairs traversing link l	\

Table 2: List of notations. The top part lists all the notations required by μFAB, while the ones in the bottom part help to illustrate the design of μFAB.

of the recursion (6)-(7) is exactly a solution to the optimization problem (4).

Figure 19 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 19a 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 19b shows that the dual control of Eqn 7 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 (6)-(7) 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 [11, 30, 33, 56]. 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 [40], 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.3 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

$$T_{ji} + T_{ij} = T_j \tag{8}$$

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

$$\frac{d}{dt} R_i(t) = \kappa \frac{R_i(t)}{\bar{T}_i} \left(1 - \frac{y_i(t)}{C_i}\right) \tag{9}$$

where

$$y_i(t) = \sum_j A_{ij} x_j(t - T_{ji}) \tag{10}$$

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

$$x_j(t) = w_j \left(\sum_i A_{ij} R_i(t - T_{ij})^{-\alpha}\right)^{-1/\alpha} \tag{11}$$

and

$$\bar{T}_i = \frac{\sum_j A_{ij} x_j(t) T_j}{\sum_j A_{ij} x_j(t)} \tag{12}$$

is the average round-trip delay over all packets passing through resource i. The equilibrium of the dynamical system (9)-(11) is the weighted α-fair allocation and is locally stable [33] if

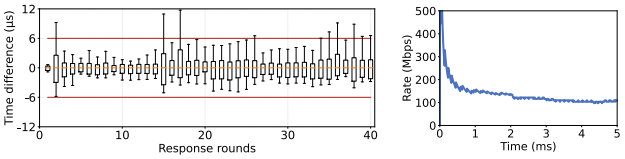
$$\kappa < \frac{\pi}{2}.$$

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 (9) 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.

D SIMULATION OF HETEROGENEOUS RESPONSE DELAYS

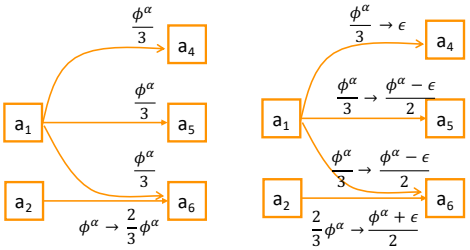
We use NS-3 simulations to evaluate the impact of asynchronous responses received by different senders on the convergence of μFAB, since a lot of ancillary work needs to be done before deploying μFAB in a real large-scale production data center. We randomly generate background traffic at 50% load and then select 128 senders and 1 receiver to generate incast traffic.

Figure 20a shows that the senders receive probe responses asynchronously, even with the time gap beyond 1 RTT (the time between the two red lines). The response time is organized into 40 rounds, that is, when the sender receives a new response, the number of response rounds increases by 1. The response time difference is calculated based on the median of the response time within the same response round. This suggests that different senders get responses



(a) Response time difference (b) Rate evolution

Figure 20: Convergence with asynchronous responses.



(a) Sufficient demand (b) Insufficient demand

Figure 21: Example of μFAB's token assignment.

asynchronously, which is consistent with the experiment phenomenon. Figure 20b shows the rate evolution of one sender. We can see that even with asynchronous responses, the sending rate converges quickly.

E 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 21a 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 21b 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 21b, 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 [45] uses the

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** $\boldsymbol{P}$;

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.

F SUPPORT MULTIPLE UNDERLAY PATHS PER VM-PAIR

μFAB 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.

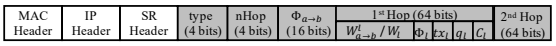
Algorithm 2 shows the overall procedure. To ensure fairness, we split the total VM-pair token equally to all paths initially (Line 3). In case some path has insufficient traffic demand, we dynamically redistribute the spare capaity to other paths for work conserving (Line 11). Nevertheless, we admit 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 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.

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

G PACKET FORMAT

Figure 22 shows the probe/response packet format of μFAB. We use standard SR header to enforce source routing, followed by our customized INT header. 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.



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.
$\Phi_{a\to b}$ (24 bits) : the sender-side (or receiver-side) 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.

Figure 22: The probe/response packet format of μFAB.

H THE RESOURCE CONSUMPTION OF μFAB'S IMPLEMENTATION

Modules	LUT(%)	Registers(%)	BRAM (%)	URAM(%)
Packet Scheduler	0.8%	1.1%	0.8%	5.7%
Context Tables	0.2%	0.2%	4.6%	3.1%
Path Monitor	0.9%	0.7%	4.8%	0.6%
TX/RX pipes	0.3%	0.1%	1.2%	0.0%
Vendor Modules	5.5%	3.6%	5.0%	0.0%
Total	7.6%	5.8%	16.4%	9.5%

Table 3: μFAB-E's hardware resource consumption based on Xilinx Alveo U200 card. Each number indicates the percentage of resources consumed for the corresponding type.

Resource Type	20K	40K	80K
Match Crossbar	8.64%	8.64%	8.64%
SRAM	17.29%	17.71%	18.75%
TCAM	6.25%	6.25%	6.25%
VLIW Actions	18.23%	18.23%	18.23%
Hash Bits	17.03%	17.05%	17.07%
Stateful ALUs	47.92%	47.92%	47.92%
Packet Header Vector	20.05%	20.05%	20.05%

Table 4: For different numbers of VM-pairs, resource consumption of μFAB-C prototype on Intel Barefoot Tofino. Each number indicates the percentage of resources consumed for the corresponding type.