IntFlow: Integrating Per-packet and Per-flowlet Switching Strategy for Load Balancing in Datacenter Networks

Qingyu Shi, Fang Wang, Dan Feng, *Member, IEEE, ACM,*

***Abstract*—Datacenter network load balancing schemes handle network traffic generated by massive different applications. Some packet-based or flowlet-based schemes capture traffic bursts for load balancing. But frequent rerouting within a flow can mix ACKs belonging to different paths in congestion control protocols, which adversely affects flow rate control. Besides, performance optimization effect of flowlet-based schemes may be less noticeable under smoother workloads. And several packet-based mechanisms implemented at end hosts can proactively reroute congested flows based on flow status even under a smooth workload, but fail to improve performance with the bursty nature of traffic. Therefore, existing schemes cannot adapt to different burst levels of dynamic traffic in datacenter networks and still have significant performance flaws in some ways.**

This paper proposes IntFlow, a novel load balancing scheme that integrates end-host based per-packet monitoring of flow status with flowlet switching in programable switches. IntFlow proactively reroutes flows experiencing network congestion or failures and avoids doing flowlet switching for small flows with high sending rate. IntFlow can provide excellent performance under both high burst and smooth workloads. Finally experimental results show IntFlow achieves up to 32% and 28% better performance than CONGA and Hermes under asymmetries, respectively.

***Index Terms*—datacenter networks, load balancing, programmable switches**

I. Introduction

THE increasing performance demands from datacenter applications (e.g. big-data analytics, web services, cloud storage and other cloud applications) pose a great challenge for datacenter networks. Datacenter networks should provide large bisection bandwidth and low latency for all different applications. To achieve this, datacenter networks typically provide multiple paths between host pairs to support load balancing, where multi-rooted topologies, such as fat-tree and leaf-spine topologies, are widely deployed. Balancing traffic among these different routing paths can greatly improve performance for bandwidth- and latency-sensitive datacenter applications[1], [2], [3].

As the standard load-balancing scheme in today's datacenters, Equal Cost Multiple Path (ECMP)[4], randomly distributes flows among multiple paths according to a hash function using certain tuples from the packet header. ECMP is widely implemented because it is readily deployed with standard unmodified TCP/IP stacks and commodity datacenter switches. However, ECMP can cause significant performance degradation because of hash collisions and the lack of adaptability to network conditions[5]. Therefore, many excellent solutions have emerged to achieve better load balancing.

In order to make best use of network resources, load balancing schemes need to distribute traffic evenly to multiple paths. Some mechanisms (e.g. Hedera[2], MicroTE[6]) optimize the path selection with a centralized traffic monitor. But because of long scheduling intervals, they are not adaptive to the traffic volatility of datacenter networks, and this is a fatal flaw for latency-sensitive applications. In contrast, many schemes (e.g. DRB[7], RPS[8], Presto[9], DRILL[10]) implemented in end hosts or switches spray the fixed switching units, including per-packet and per-flowcell, across available paths to make full use of link resource. They are prone to experience packet reordering under asymmetric topology[5], [11]. Some other schemes (e.g. CONGA[12], HULA[13], CLOVE[14]) can split flows into flowlets[15] for load balancing. As we know, flowlets are bursts of packets from a flow, which are separated by large enough gaps. Flowlets will not cause much packet reordering when they are rerouted on different paths at an appropriate time interval. However, flowlets are decided by traffic characteristics, which are affected by many factors such as applications and transport protocols. Therefore schemes relying on flowlet switching are inherently passive and cannot always timely react to congestion when the workload is smooth. Besides, all schemes mentioned above always vigorously reroute flows when encountering a fixed switching unit or a new flowlet, which leads to too frequent rerouting under heavy loads, where the possibility that small flows are split into finer-grained units could cause reordering and increased latency. Another drawback is that frequent rerouting within a flow can mix ACKs belonging to different paths in congestion control protocols, which adversely affects flow rate control because congestion control algorithms consider all ACKs as congestion feedback signals from the current path to adjust the rate (window) of a flow. This phenomenon is called congestion mismatch, which is first unveiled in [5].

Manuscript received June 25, 2019; revised January 19, 2020; accepted April 23, 2020.

This work was supported by National Defense Preliminary Research Project No. 31511010202, NSFC No. 61832020, No. 61821003, Fundamental Research Funds for the Central Universities, CERNET Innovation Project NGII20170120. *(Corresponding author: Fang Wang.)*

The authors are with the Key Laboratory of Information Storage System (School of Computer Science and Technology, Huazhong University of Science and Technology), Ministry of Education of China, Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China, and F. Wang is also with the Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen 518000, China. (e-mail: qingyushi@hust.edu.cn; wangfang@hust.edu.cn; dfeng@hust.edu.cn).

According to the analysis above, passive, frequent and vigorous rerouting within a flow can cause unpredictable packet reordering and congestion mismatch problem, which can degrade performance. Thus, several solutions (e.g. FlowBender[16], Hermes[5]) perform proactive load balancing based on **flow status, which includes the remaining flow size, the sending rate, the congestion state of flow, etc.** FlowBender reroutes flows whenever congestion is detected by end hosts. but its random and blind rerouting brings sub-optimal performance under high loads. Hermes detects flow conditions at end hosts, and reacts by timely yet cautious rerouting at packet granularity. However, Hermes cannot accurately calculate the cost of congestion mismatch caused by rerouting, because some metrics such as the flow sending rate after rerouting on the new path cannot be effectively measured. Therefore some good rerouting opportunities may be ignored without flowlet switching. For example, a large flow can be transmitted on multiple uncongested paths using flowlet switching, while Hermes may believe that this flow should be smoothly transmitted at the current path in a larger time range than the flowlet timeout. In a word, this kind of schemes do not rely on flowlet switching and ignores the effect of traffic bursts on improving load balancing performance.

According to our observation, we find that both sufficient rerouting opportunities and cautious rerouting decisions based on flow status are extremely important for load balancing. We ask the flowing question: Can we achieve the above two aspects at the same time to adapt to dynamic traffic in datacenter networks? In this paper, we present IntFlow to answer this question.

IntFlow assesses flow status at end hosts to assist the flowlet switching at programmable switches (e.g. Barefoot Tofino[17], Intel's FlexPipe[18]), which are able to parse packet headers, match custom fields in headers and perform corresponding actions. IntFlow makes use of events (e.g. retransmissions and timeouts), the flow sending rate, the remaining flow size, which can be estimated with the size sent, and the accurate latency of current path to judge flow status. The senders at end hosts mark the congestion status of flows on the field of encapsulation header with different values based on flow status in overlay networks[19]. When the flow passes through the switches, IntFlow exploits the capability of programmable switches to make deliberate rerouting decisions based on the congestion mark of the flow and flowlet gaps. In this way, IntFlow can proactively reroute flows timely once it senses congestion or failures to make up for the inadequacies of traditional flowlet switching schemes. Furthermore, IntFlow reroutes new flowlets cautiously without causing the congestion mismatch problem. In summary, the contributions we make are three-fold:

- We empirically analyze the limitations of current load balancing schemes and find that no scheme takes into account both sufficient rerouting opportunities and cautious rerouting decisions based on flow status.
- We present IntFlow, a novel load balancing solution, which integrates per-packet and per-flowlet switching in the network. IntFlow reacts to congestion and failures timely based on flow status to achieve proactive rerouting, while performing cautious rerouting for flowlet switching.
- We evaluate IntFlow via large-scale NS3[20] simulations, showing that IntFlow achieves up to 32% and 28% better performance than CONGA and Hermes under asymmetries, respectively.

In the remainder, supported by an empirical study, we show the background and our motivation in more detail in section II. Then we present our mechanism IntFlow in section III. We evaluate IntFlow and show the superiority of IntFlow compared to other solutions in section IV. We discuss some design consideration and potential deployment concerns in section V. Finally we briefly introduce the important related work in section VI and summarize our work in section VII.

II. Background and Motivation

Current datacenter contains a complex network environment, which provides several alternative routing paths between any two end-hosts which are connected by different switches. Datacenter networks should provide high bandwidth and low latency to meet the performance requirement of a wide variety of datacenter applications. Modern datacenter networks mainly have the following two features:

- **Traffic dynamics:** Datacenter networks typically have dynamic traffic[21], [22], [23], [5], where applications which are sensitive to bandwidth (e.g. MapReduce) and sensitive to flow completion time (e.g. Memcached) coexist. And both smooth and bursty network traffic may occur in any time.
- **Asymmetries:** Network asymmetry is common for modern datacenters in practice[24], [5]. For example, different paths between one or more source/destination pairs have different amounts of available bandwidth. This is Because the datacenter evolvement adding racks and switches can cause coexistence of heterogenous switches and link or switch failures can also create asymmetries. Besides, switch failures such as random packet drops can also induce topology asymmetry[23], [5].

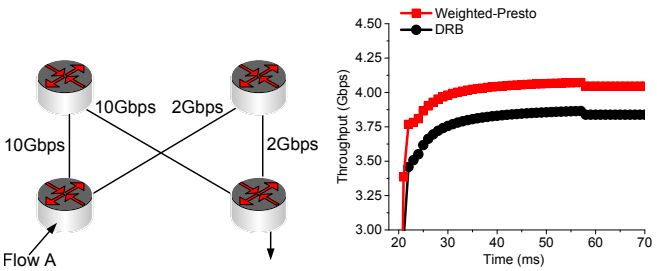
Balancing load adaptively under above situation is a significant challenge. However, current optimized solutions still have shortcomings. Flowlet switching has been found effective without causing much packet reordering for load balancing over asymmetric (e.g. with different available bandwidth) paths[24]. Many schemes leverage flowlet switching to achieve fine-grained load balancing, such as CONGA[12], CLOVE[14] and LetFlow[24]. However, The pure flowlet switching has two shortcomings. The first is that flowlet switching cannot timely react to congestion by splitting flows under smooth traffic. This is because network traffic in datacenters can be unpredictable with massive different applications. Besides, DCTCP[25] is less bursty than TCP, which is not conducive to generating more sufficient inactivity gaps that form flowlets. And under heavy bursty traffic, end hosts or network devices can observe a large number of new flowlets. This creates the second shortcoming, where frequent and vigorous rerouting in flowlet switching causes congestion mismatch problem. The problem refers to that rerouting events can cause a mismatch between the sending rate and the state of the new path because

congestion control algorithms adjust the rate (window) of a flow based on the congestion state of the current path. Because new flowlets are always switched to the best available path whenever they are captured in previous schemes, this problem hinders the utilization of link bandwidth especially under asymmetric topologies. For example, when rerouting events happen with ECN-based congestion control protocols, outdated ACKs with no ECE mark of the other path may improperly increase the sending rate (window), while the ones with an ECE mark will mistakenly decrease the sending rate. And schemes that frequently reroute fixed switching units, including packets, flowcells and flowlets, among different paths but do not considering flow status may also cause congestion mismatch, such as RPS[8], DRILL[10], Presto[9], CONGA, etc.

Considering problems above, Hermes[5] employs a cautious rerouting algorithm with per-packet switching based on flow status and path conditions at end hosts. But it abandons the flowlet switching, which means it can not take advantage of the bursty nature of traffic. [5] shows that though Hermes outperforms other flowlet-based schemes under a smooth workload, under a bursty workload its performance is surpassed by multiple solutions using flowlet switching in an asymmetric topology. This is most likely because only relying on monitoring flow status for load balancing may lose some good rerouting opportunities. Similarly, FlowBender[16] keeps a watch on congestion signals for every packet received at end hosts for proactive rerouting. However, it still dose not explore the performance improvement with bursty workloads. And FlowBender brings sub-optimal performance because of random rerouting for load balancing.

Therefore, though many prior schemes split a flow to a fixed switching unit, which includes per-packet, per-flowcell and per-flowlet, provide sufficient rerouting opportunities, they have the risk of causing packet reordering and congestion mismatch problems, such as RPS, DRILL, Presto, etc. Besides, performance optimization effect of flowlet-based schemes may be less noticeable under smoother workloads. On the contrary, though schemes rerouting flows at packet granularity based on flow status can make proactive and cautious load balancing decisions, they lose the rerouting opportunity based on traffic bursts. That is, current mechanisms have drawbacks in handling dynamic traffic and asymmetric topologies in datacenters, which incurs a significant performance penalty. The performance loss caused by these problems is experimentally demonstrated below.

At first, in order to quantify the impairment of congestion mismatch problem in load balancing, we implement DRB[7] and a variant of Presto[9] under an asymmetric topology in NS3. DRB splits traffic to different routing paths at packet granularity, while Presto leverages flowcells to distribute traffic proportionally to path capacity. As we analyzed above, DRB and Presto can result in congestion mismatch because of frequent and vigorous rerouting in load balancing. We use a simple 2×2 leaf-spine topology and an asymmetric network with 2 and 10 Gbps paths shown in Fig. 1a. To mask the throughput loss caused by packet reordering, we implement a reordering buffer and set **DupAckThreshold** to 1000. DCTCP is used as the default transport protocol like prior load balancing schemes.



(a) An asymmetric network (b) Throughput of flow A

Fig. 1: The congestion mismatch problem causes severe throughput loss.

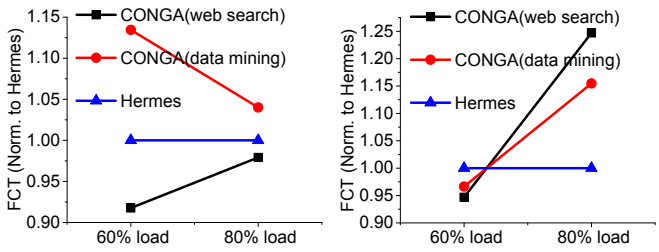
As the Fig. 1b shows, flow A only achieves a throughput of around 4 Gbps in DRB and Presto. And Presto spreads flowcells using 2:10 ratio to match path capacities in our implementation. The results are similar to [5]. This is because when rerouting events happen, the congestion feedback of ACKs belonging to the path with 2 Gbps bandwidth (the right path) constrains the congestion window, which causes the throughput loss in the path with 10 Gbps bandwidth (the left path). Similarly, the congestion feedback from the right path can cause incorrect adjustment of congestion window in the left path, which may lead to sudden congestion because the left path cannot immediately absorb such a traffic burst. As a result, such a congestion mismatch causes throughput loss.

TABLE I: The frequency of rerouting under different workloads with 80% load level.

Workloads	Schemes	Small flows	Medium flows	Large flows
Web-search	Hermes	260	69349	44990
	CONGA	107603	3473187	1110189
Data-minining	Hermes	947	604	35839
	CONGA	6	32	11

We next quantify the shortcomings of different load balancing mechanisms when they handle different application workloads. We implement CONGA and Hermes in NS3 based on two widely-used realistic workloads (web-search[25] and data-mining[3]) observed from deployed datacenters in a 8×8 leaf-spine topology with 10 Gbps links and 128 servers under asymmetry. The web-search workload is much more bursty than the data-mining workload. CONGA employs per-flowlet switching in the network and thus it makes full use of traffic bursts to get a lot of rerouting opportunities. Hermes employs per-packet switching in end host and achieves proactive rerouting based on flow status.

At first, we count the number of times the flows with different size change paths to show the frequency of rerouting. The results are shown in Table I. We can find that CONGA changes the paths of flows much more often than Hermes under the web-search workload, while under the data-mining workload Hermes performs rerouting more frequently. And



(a) Overall avg FCT (b) Small flow(<100KB) avg

Fig. 2: FCT statistics in the asymmetric topology(normalized to Hermes).

flow completion time (FCT) is used as the primary performance metric in Fig. 2, where we normalize the FCT to Hermes in order to better visualize the results. As Fig. 2a shows, under the web-search workload CONGA outperforms Hermes by 8%, while Hermes performs over 10% better than CONGA under the data-mining workload in the results. This is because CONGA exploits the bursty arrivals of traffic to get more rerouting opportunities under the web-search workload, while Hermes not waiting for the flowlet timeout proactively reroutes flows. But traffic in production datacenters can be highly dynamic. Besides, as Fig. 2b shows, the average FCTs for small flows grow dramatically for CONGA as the load increases in our simulation. As we can see in Table I, rerouting times for small flows in CONGA are hundreds of times that in Hermes under the web-search workload. This is because too many small flows are broken into new flowlets, and thus they are heavily affected by packet reordering and congestion mismatch. Therefore, flowlet switching needs a cautious rerouting mechanism to avoid unnecessary path switching for small flows. All in all, both CONGA and Hermes cannot achieve adaptive performance for dynamic traffic. This inspires us to consider how to provide an adaptive rerouting mechanism for the complex network environment by combining the two technical advantages of monitoring flow status and flowlet switching.

According to our observation, applying flowlet switching in the network has been already a mature technology[12], [13], [24], [26]. In-network flowlet switching can timely capture traffic bursts and achieve sufficient visibility for making appropriate load balancing decisions. However, the tremendous amount of flows in datacenter networks makes it difficult to monitor flow status in network devices, while end hosts provide sufficient computing and storage resources to accomplish this[5]. Thus we attempt to monitor traffic status at end hosts and pass some information about flow status to network devices. We find current new-generation programmable switches (e.g. Barefoot Tofino, Intel's FlexPipe) can help. They allow users to program the switch packet-processing pipeline without replacing the switch ASICs(e.g. P4[27]). As the performance and capability of programmable switches make them appealing to be used beyond traditional network functionalities, many solutions for in-network computing, caching and load balancing (e.g. Eris[28], NetCache[29], DistCache[30], HULA, CLOVE-INT) have emerged. We can also use programmable switches

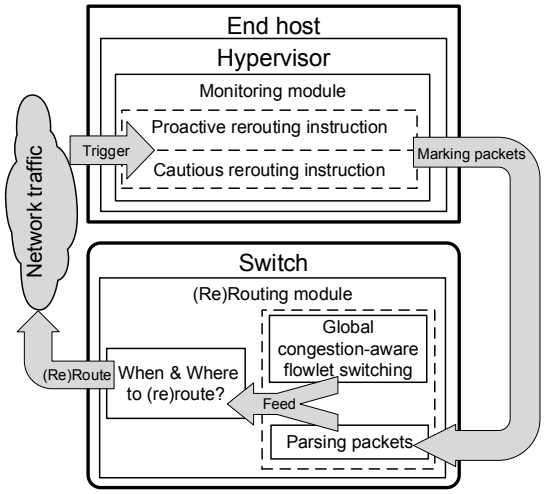


Fig. 3: IntFlow overview.

to design a novel in-network load-balancing scheme to achieve our goal. In addition to applying the flowlet-based load-balancing strategy in the network like previous studies[13], [26], we also define and parse a certain field of packet header in the programmable switch, which contains the information about flow status and helps us perform proactive and cautious rerouting in the network. We detail our design in the next section.

III. Design

A. Overview

We propose IntFlow, a novel scheme that integrates per-packet and per-flowlet switching strategy for load balancing. It exploits programmable switches to perform proactive and cautious rerouting in the network. Fig. 3 overviews the two main modules of IntFlow: (1) the monitoring module that is responsible for monitoring flow status and marking packets; and (2) the rerouting module that is responsible for making deliberate load balancing decisions. With these two modules IntFlow can proactively reroute flows which are experiencing congestion or failures, and prevent small flows with a high sending rate being switched to other paths even though encountering new flowlets to mitigate performance degradation due to congestion mismatch and packet reordering. Besides, IntFlow reroutes new flowlets carefully to make full use of the bursty nature of traffic. It distributes flowlets within the emerging programmable switches to achieve best visibility into the network.

B. The Monitoring Module

The monitoring module leverages the condition sensing algorithm to perform proactive and cautious instructions. It firstly discovers congested flows and healthy small flows that do not need to be rerouted at packet granularity, and then marks packets with different values on the encapsulation header based on the overlay network to indicate different decisions, which are used to assist per-flowlet switching at

Algorithm 1: Condition sensing

Input: f : every flow generated by applications
Output: IN : rerouting $instruction$ from the monitoring module

```
1  for every packet do
2      IN = 0;                    /* Initialization */
3      if f.packet is a retransmitted packet then
4          IN = 4;
           /* Marked as a retransmission
           packet                                   */
5      end
6      if f is not a new flow then
7          if f.timeout > N_timeout or f.retransmission >
             P_retrans then
8              IN = IN | 1;
               /* Should be rerouted               */
9          else
10             if f.delay ⩾ T_Delay_high then
11                 if f.size ⩾ S and f.rate ⩽ R then
12                     IN = IN | 2;
                       /* Should be rerouted       */
13                 end
14             else
15                 if f.size < S then
16                     IN = IN | 3;
                       /* Should not be rerouted
                       */
17                 end
18             end
19         end
20     end
21     return IN;
22 end
```

programmable switches. The flow status is marked with a 3-bit value, which is the value of IN in Algorithm 1.

Every packet generated by applications in datacenters will go through the monitoring module and get marked by the condition sensing algorithm, which is shown in Algorithm 1. Firstly, when datacenter networks experience failures (e.g. switch and link failures or malfunctions) or severe congestion, many packets will not reach the destination host on time or be dropped in the network. The highest bit of IN is used to indicate whether the packet is retransmitted. And thus the value of IN is marked as 4 for all retransmitted packets (lines 2-5). In our simulation we observe that this kind of packet drops can trigger TCP retransmissions. So IntFlow calculates flow retransmission rates (e.g. 1% for the $P_{retrans}$ referring to previous study[5]) for every few milliseconds and records the number of timeouts (e.g. 3 for the $N_{timeout}$) to selectively and proactively reroute flows (lines 6-8). The value of IN will be set to $IN \mid 1$ to indicate that rerouting is required because of network failures or severe congestion.

Besides, IntFlow proactively and effectively reroutes flows that are experiencing network congestion. In addition to that the load characteristics in datacenters may be stable in many cases (e.g. in big-data analytics), DCTCP also makes the flow sending rate smoother. DCTCP always uses an estimate α to resize it's window size smoothly:

$$cwnd = cwnd \times (1 - \alpha/2) \tag{1}$$

Thus, pure flowlet switching cannot always timely react to congestion in datacenter networks. Whether a rerouting event can improve the flow's completion time depends on many factors, such as the remaining flow size and the sending rate. The packet reordering and congestion mismatch[5] due to rerouting events can reduce the sending rate at first. Then the sending rate on the new path will gradually increase. Therefore, if the current sending rate is already high, the rerouting may be not benefited. Moreover, frequently rerouting a flow with a small remaining size may have limited benefit.

Considering the above situation, we perform proactive rerouting based on flow status, which includes the flow sending rate, the remaining flow size and one-way delay of the flow. To achieve this, we use the size a flow already sent to estimate the remaining size[31], the CONGA's DRE algorithm to measure the sending rate and one-way delays to distinguish whether the flow is experiencing congestion. As the lines 9-13 in Algorithm 1 shows, IntFlow proactively reroutes a flow that is experiencing congestion (with a high one-way delay over T_{Delay_high}). This rerouted flow should has an appropriate remaining size (S) and its sending rate is not very high (R). The value of IN will be set to $IN \mid 2$ to indicate that rerouting is required because of general network congestion. We set T_{Delay_high} (180μs in our simulations) to be at least base RTT plus 1.5$\times$ of the one hop delay, S to be 600KB and R to be 30% of the link capacity by default referring to previous study[5]. The performance of IntFlow is fairly efficient and stable with the parameter settings in our simulations.

One-way delays which the flow experiences in the network directly indicate the extent of path congestion[32], [33], [34]. IntFlow can employ In-band Network Telemetry (INT)[35] to measure one-way delays between VMs which are connected by different switches in datacenters. INT is a technology available in programmable switches, which enables network endpoints to embed instructions in packets, requesting every network hop to insert network state in packets as they traverse the network at line rate. INT can capture network latency that the packet encounters at switches (delta between local egress timestamp and local ingress timestamp). When the data packet is received by the destination hypervisor, IntFlow can compute the one-way delay as a sum of the per-hop latencies (under the assumption that switching and queueing latencies dominate and propagation delays are minimal, which is typically true in today's networks). IntFlow records the one-way delay for the flow at the destination hypervisor and encodes it into the option fields of TCP header in the corresponding ACKs to inform the source host.

Note that although IntFlow uses a method similar to Hermes at end hosts to monitor flow status, it is completely different in design from Hermes. Hermes relies entirely on monitoring at end hosts to make load balancing decisions, but IntFlow uses a two-step decision method. IntFlow only marks flows based on flow status observed at end hosts at the first step. And then

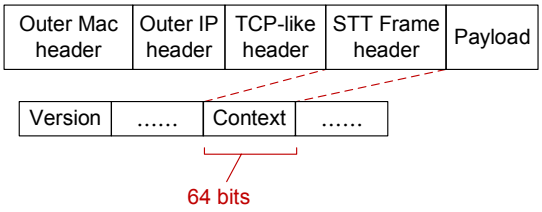


Fig. 4: Encapsulation with STT headers in IntFlow.

the information about flow status is used for load balancing in the network switch at the second step. In this way, IntFlow can explore flowlet switching opportunities in the network while ensuring the advantages of proactive rerouting in the end hosts.

Moreover, because frequent rerouting in flowlet switching may negatively affect small flows without cautiously considering the benefit of rerouting (referring to the empirical study in Section II), we avoid the uncongested small flows switching path even though new flowlets appear (lines 14-17). Here we use one-way delays (T_{Delay_high}) to determine the degree of congestion and a tuned flow remaining size (S) to define small flows. The value of IN will be set to $IN \mid 3$ to indicate that rerouting is not allowed. If the packet belongs to a new flow or a normal flow that is not marked in lines 6-20 of Algorithm 1, IntFlow will leverage new flowlets to balance load. The value of IN will be set to 0 or 4 to indicate that the flow can be rerouted when it encounters new flowlet timeout in the network.

Finally, the monitoring module will insert different values from the output IN of Algorithm 1 into the encapsulation header in network overlays, which have been recently widely adopted in multi-tenant datacenter networks[14], [34]. Network overlays leverage an encapsulation header (e.g. Stateless Transport Tunneling (STT)) to route the packet in the physical network. IntFlow exploits the STT context (shown in Fig. 4) to encode the output value IN. STT context field containing 64 bits provides sufficient fields to encode this value. In this way, the rerouting module in programmable switches can parse the value IN to identify different flow status, where the actual load balancing decision will be performed.

C. The Rerouting Module

In addition to recording network latency, the rerouting module residing in programmable switches performs actual load balancing decisions. Unlike previous flowlet-based schemes, it selectively reroutes flows when encountering new flowlets due to the traffic burst. The rerouting module parses the value IN from the encapsulation header of packets, which is encoded in a fixed format by the monitoring module to indicate the flow status. Whether to reroute flows needs to consider the flow status. Previous work has proven that the sending rate on every particular port of programable switches can be measured at an acceptable cost[13], [26]. Based the Discounting Rate Estimator (DRE)[12] algorithm in CONGA, IntFlow implements a variant of Discounting Rate Estimator (DREva) to measure the transmission capacity of every link and conveys real-time path congestion metrics to other leaf switches under the leaf-spine topology. The leaves can use these metrics to achieve a global congestion-aware view for load balancing.

Algorithm 2: Rerouting logic

Input: f : every flow passing through the leaf switches
Input: IN : rerouting $instruction$ encoded by the monitoring module
Output: $port$: the output port in the switch

```
1  for every packet do
2      if f is not a new flow then
3          if (IN&4)! = 0 then
4              update local DREva table;
5              IN = (IN&3);
6          end
7          if IN == 3 or (IN ==
            0 and f.flowlet_timeout == false) then
8              port = f.old_port;
               /* Should not be rerouted    */
9          else
10             if IN == 1 then
11                 port = Min_congestion_some_ports;
                   /* Should be rerouted to
                   the least congested path of
                   some specific paths       */
12             else
13                 port = Min_congestion_all_ports;
                   /* Should be rerouted to
                   the least congested path of
                   all paths                 */
14             end
15         end
16     else
17         port = Min_congestion_all_ports;
           /* Choose the least congested
           path                              */
18     end
19     return port;
20 end
```

TABLE II: Design logic of IntFlow

IN&3	Flow status (at end hosts)	Whether to reroute (at programable switches)
0	unassigned state	yes only when new flowlets appear
1	failures or severe congestion	yes (reroute in some specific paths)
2	general network congestion	yes (reroute in all paths)
3	uncongested small flows	no

IntFlow takes into account both flow status and traffic characteristics for load balancing through per-packet monitoring at end hosts and per-flowlet switching at network. To better illustrate the design logic, we summarize it with Table II. Next, we analyze the load balancing process according to Table II and Algorithm 2.

Algorithm 2 shows the rerouting logic at the programable switch. It is timely triggered for every packet to check the

flow status and the flowlet table. IntFlow can detect packet dropping in the network according to the value of IN because the retransmitted packets are marked in the highest bit of IN. Thus, the transmission capacity of every link can be estimated with network utilization and degree of network packet dropping. The DREva algorithm maintains a register, X, which is incremented for each packet sent over the link by the packet size in bytes. If the packet is retransmitted (lines 3-6), X for the retransmitted link is also incremented by β times the packet size. X is decremented periodically (every T_{dreva}) with a multiplicative factor between 0 and 1, but if the retransmitted packet occurs, the next periodic X value reduction will become less, after which it will return to normal. It is easy to show that under normal conditions without packet dropping, X is proportional to the rate of traffic over the link. When some switches failures such as random packet drops occur under a symmetric network topology, X is likely to have a larger value on the link with a higher packet dropping rate. DREva here uses a similar method to the DRE algorithm to obtain the congestion metric of the link by quantizing $X/C\tau$ to 3 bits (C is the link speed and τ is a tuned parameter). But DRE does not consider the impact of packet dropping in the network, which causes severe performance loss under the scenario of random packet drops in our evaluation.

As lines 7-9 in Algorithm 2 shows, for the uncongested small flows (the value of IN is 3) and flows in unassigned state (the value of IN is 0) not reaching the flowlet timeout, IntFlow does not reroute them. On the contrary, those pure flowlet-based schemes (e.g. CONGA, LetFlow) may vigorously reroute small flows as long as the flowlet times out.

Besides, based on the global congestion awareness the flow is rerouted to the least congested path of some specific paths, which are other available paths except the current forwarding path , when the value IN of the flow is 1 (Algorithm 2 lines 9-10). This is because according to the flow status, IntFlow believes that the current path has a high probability of network failure and severe network congestion, thus the flow is rerouted to other paths as much as possible. When the flow is in unassigned state and encounters a new flowlet (the value of IN is 0 and the flowlet times out), or is experiencing general network congestion (the value of IN is 2), it is rerouted to the least congested path of all paths (lines 12-13). And if the flow is observed as a new flow at the switch, it is forwarded to the least congested path of all paths (lines 16-17). In a word, IntFlow can proactively reroute the flows suffering from failure/timeout and congestion, and also properly leverage rerouting opportunities of flowlet switching.

To summarize, IntFlow achieves proactive and cautious load balancing by integrating end-host based per-packet monitoring of the flow status with flowlet switching in programable switches.

IV. Evaluation

We evaluate IntFlow compared with the state-of-the-art load balancing schemes to study the performance improvement in large-scale simulations. All experiments are implemented on the discrete-event network simulator NS3[20]. Our evaluation seeks to show whether IntFlow can perform optimally under traffic dynamics and asymmetries in datacenter networks.

Topology: We build a 8×8 leaf-spine topology with 10Gbps links and 128 servers with NS3. There are 8 equal cost paths between any pair of hosts that are connected by different switches. Therefore we simulate a 2:1 oversubscription at the leaf level to meet the typical deployment of current datacenter networks[12].

Workloads: We use two widely-used realistic workloads (web-search[25] and data-mining[3]) observed from deployed datacenters to provide traffic dynamics for evaluations. As shown in previous work[12], [13], [5], [24], both of these workloads are heavy-tailed and most flows are small, but the small fraction of large flows contributes to a great portion of total bytes. And the web-search workload is more bursty, while the data-mining workload is more skewed with 95% of all data bytes belonging to ∼3.6% of flows that are larger than 35MB[12]. Thus, it is more challenging for flowlet-based schemes to balance load under the data-mining workload. We generates flows between random senders and receivers under different leaf switches according to Poisson processes with varying traffic loads.

Metrics: Similar to previous work, we use flow completion time (FCT) as the primary performance metric. In addition to the overall average FCT, we also take the FCT for small flows (<100KB) and large flows (>10MB) into consideration for better understanding of performance. And the 99th percentile FCT for small flows is also an important performance metric.

Methodology: In order to show the performance gains of IntFlow, besides ECMP we compare IntFlow with the following state-of-the-art schemes using DCTCP[25] as the default transport protocol:

- **LetFlow.** LetFlow simply picks a random path for each flowlet at network switches.
- **CONGA.** The scheme CONGA employs global utilization-aware flowlet switching at network switches.
- **CLOVE-ECN.** CLOVE-ECN leverages ECN-based feedback to route flowlets at end hosts. Because [14] has shown that CLOVE-INT is outperformed by CONGA, we do not simulate CLOVE-INT.
- **Hermes.** Hermes leverages its comprehensive sensing to detect path conditions for every packet at end hosts, and it reacts using timely yet cautious rerouting but does not wait for flowlets.

Note: We do not show FCTs of all previous solutions in our simulations, which includes MPTCP, Presto, DRILL, etc. MPTCP[36] shows performance instability and worse performance gains compared with CONGA in many scenarios. Presto[9] and DRILL[10] suffering from packet reordering and congestion mismatch fail to provide performance beyond CONGA or Hermes under high asymmetries[5], [11]. We adopt a consistent flowlet timeout value (e.g. 250μs) in all schemes for fairness. If the flowlet timeout value is set too small, flowlet-based solutions cause serious performance losses because of congestion mismatch under asymmetry. But because DCTCP is less bursty than TCP, the default 500μs flowlet timeout value in CONGA is too big. Therefore we try

several different flowlet timeout values and adopt the most appropriate one (250μs) in our simulations.

A. Asymmetries Caused by Different Link Speeds

To compare IntFlow with other schemes under an asymmetric topology, we reduce the link capacity from 10Gbps to 2Gbps for 20% of randomly selected leaf-to-spine links. We normalize the FCT to IntFlow in order to better visualize the results.

As we analyze in section II, frequent rerouting based on per-flowlet switching causes degraded performance and proactive rerouting based on flow status can lose many rerouting opportunities. And IntFlow achieves proactive and cautious load balancing decisions to solve above problem. Therefore, at first we verify whether IntFlow can provide appropriate rerouting frequency under different application workloads.

TABLE III: The frequency of rerouting flows.

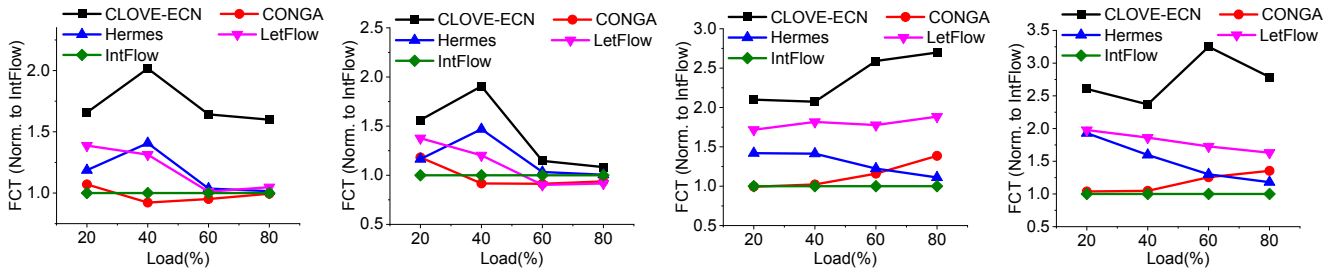
Workloads	Schemes	Small flows	Medium flows	Large flows
Web-search	Hermes	260	69349	44990
	CONGA	107603	3473187	1110189
	IntFlow	65381	3351111	1033257
Data-minining	Hermes	947	604	35839
	CONGA	6	32	11
	IntFlow	747	920	26582

Table III shows the frequency of rerouting flows in different schemes under the web-search and data-mining workload at 80% load level. For the web-search workload, IntFlow has much more rerouting times than Hermes to explore performance gains. IntFlow achieves cautious rerouting to avoid unnecessary rerouting events that degrade performance, and thus it has smaller rerouting frequency than CONGA. Especially for the small flows under the web-search workload IntFlow reduces rerouting times by 40% compared to CONGA. CONGA can cause high FCTs for small flows at high loads due to frequent rerouting[5], while IntFlow does not reroute the uncongested small flows. For the data-mining workload, which is much smoother than the web-search workload, IntFlow obtains more good rerouting opportunities than CONGA in Table III. Because IntFlow makes proactive load balancing decisions at end hosts while CONGA always passively waits for new flowlets to reroute. Compared to Hermes, IntFlow has different rerouting frequency for flows with different size under the data-mining workload due to different load balancing mechanisms. In summary, IntFlow does not reroute small flows frequently under high load level, but it can make best use of traffic bursts to reroute flowlets. And IntFlow can obtain sufficient load balancing opportunities under smooth workload (data-mining) through proactive rerouting. Next we will expose the performance benefits of IntFlow.

Under the web-search workload: As shown in Fig. 5, IntFlow achieves almost the same performance compared to CONGA (within a performance gap of -8% to 6%), and outperforms other schemes. When loads become heavy, all schemes achieve similar performance except CLOVE-ECN. This is because the web-search workload contains many small flows and is also high bursty. These schemes employing in-network flowlet switching can quickly converge to a balanced load with enough flowlets. And Hermes leverages congestion-aware per-packet switching to improve performance. But because IntFlow can make most of good rerouting opportunities from flowlet switching and proactively reroute flows experiencing congestion or failure, at 20-40% load it performs 24-28% and 15-28% better than LetFlow and Hermes, respectively. Moreover, IntFlow outperforms CLOVE-ECN by up to 50%. CLOVE-ECN leverages ECNs to sense path congestion and is implemented at end hosts. It obtains worse visibility than IntFlow. And LetFlow is a congestion-oblivious scheme using in-network random-hashing flowlet switching. It is similar to CONGA and cannot always proactively react to congestion. Compared to Hermes, IntFlow improves FCTs for both large and small flows. This shows that IntFlow can not only proactively reroute flows based on flow status, but also take advantage of flowlet switching at the right time.

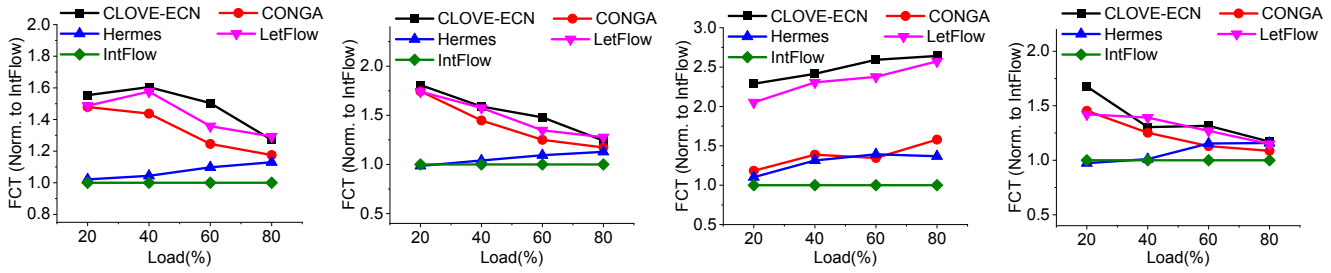
Those flowlet-based schemes can negatively affect small flows with absolute rerouting. As we can see in Fig. 5c and 5d, the average and the 99th percentile FCTs for small flows grow as the load increases or stay high in most flowlet-based solutions. Compare with other flowlet-based schemes, at 80% load IntFlow improves the average and the 99th percentile FCTs for small flows by 28-62% and 26-62%, respectively. This is because small flows are broken into several flowlets under high loads. This causes packet reordering and congestion mismatch problems for these pure flowlet-based schemes. And IntFlow monitors flow status to make cautious decisions. In IntFlow small flows observed with no network congestion will not be rerouted, while the ones experiencing congestion and encountering new flowlets will be rerouted. Therefore, IntFlow prevents some normal small flows from being rerouted and selectively reroutes other congested small flows. In this way, IntFlow alleviates congestion mismatch problem by reducing unnecessary flow switching. And Hermes solves congestion mismatch to some extent using comprehensive sensing and cautious rerouting for per-packet as the Fig. 5c and 5d show, where Hermes only performs worse than IntFlow. But IntFlow still achieves better average FCTs for small flows than Hermes by 10-29%. This is because IntFlow exploits good rerouting opportunities from flowlet switching besides achieving cautious rerouting.

Under the data-mining workload: As shown in Fig. 6, IntFlow outperforms all other schemes in most cases. IntFlow achieves 15-32% better performance than CONGA. This is because the data-mining workload contains more large flows and is significantly less bursty. Previous flowlet-based schemes (including CLOVE-ECN, LetFlow and CONGA) cannot timely react to path congestion because there are not enough flowlet gaps under workloads with low bursty. IntFlow leverages the monitoring module at end hosts to proactively reroute flows when sensing congestion/failure. Besides, because IntFlow reroutes new flowlets at the right time to improve performance compared to Hermes, IntFlow performs 2-11% better than Hermes. For the average FCTs for small flows and overall 99th percentile FCTs, IntFlow always achieves the best performance as the Fig. 6c and 6d show.



(a) Overall avg FCT (b) Large flow (>10MB) avg (c) Small flow (<100KB) avg (d) Small flow 99^{th} percentile

Fig. 5: FCT for the web-search workload in the asymmetric topology (normalized to IntFlow).



(a) Overall avg FCT (b) Large flow (>10MB) avg (c) Small flow (<100KB) avg (d) Overall 99^{th} percentile

Fig. 6: FCT for the data-mining workload in the asymmetric topology (normalized to IntFlow).

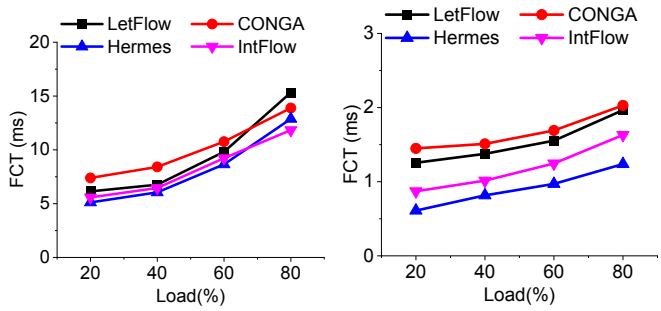
This is because IntFlow proactively reroute those large flows experiencing congestion in the data-mining workloads, and cautiously reroute small flows to avoid congestion mismatch.

Finally we also implement an idealized variant of Presto[9], which provide best-case performance for Presto. It employs per-packet load balancing and a reordering buffer to put all packets of every flow in order. Because its performance is very unsatisfactory under asymmetries, we do not compare it with other schemes in the figure. We take into account the path asymmetry by using static weights (based on the topology) to make load balancing decisions[24]. However, Presto causes high FCTs under asymmetry and does not achieve comparable performance to IntFlow. This is because congestion mismatch problem. When load gets heavy under asymmetry, the congestion window in Presto is constrained by the most congested path and the flow rate is adjusted in a chaotic way.

B. Asymmetries Caused by Random Packet Drops

As switch failures such as random packet drops can also induce topology asymmetry[23], [5], we simulate the silent random packet drop scenario to inspect the performance of IntFlow, where we set the drop rate to 2% on a randomly selected core switch. Fig. 7 shows the performance of different schemes under the web-search workload.

At first, we can observe that IntFlow achieves 14-24% better performance than CONGA in Fig. 7a. This is because flows traversing the failed switch tend to have a low sending rate due to frequent packet drops and CONGA always shifts more traffic to the failed paths because it senses and balances load based on network utilization according to the DRE



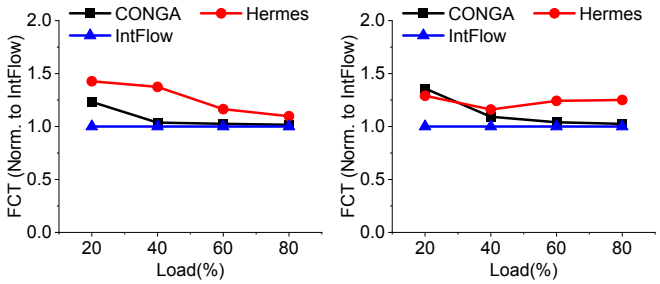
(a) Overall avg FCT (b) Small flow (<100KB) avg

Fig. 7: Performance with random packet drops (web-search).

algorithm. LetFlow is comparatively less affected because of the random rerouting strategy. IntFlow leverages the DREva algorithm to choose the best rerouting path in the network, which calculates the least congested path based on network utilization and retransmitted packets. Therefore, IntFlow can sense excessive retransmitted packets to prevent too much traffic from flowing through the failed switch. But IntFlow is still within a performance gap of -9% to 8% compared with Hermes. This is because Hermes does not capture flowlets to reroute flows under the high burst workload. Hermes can sense the switch failures and avoid routing through the failed switch. IntFlow relies on flowlet switching for load balancing, and uses network utilization for congestion estimation. Fig. 7b shows that Hermes achieves the best performance for the average FCTs for small flows and IntFlow obtains 19-39% better performance than CONGA.

C. Different transport protocols

We fnally check the performance of IntFlow with TCP under the same situation under asymmetries caused by different link speeds in above simulation. As the Fig. 8 shows, IntFlow obtains the best performance. Since TCP is more bursty, we set the flowlet timeout to be 500μs for CONGA and IntFlow as suggested in previous work[12], [5], [24]. Under the web-search workload, IntFlow outperforms Hermes by up to 30% and performs almost identically to CONGA under heavy loads. Under the data-mining workload, IntFlow achieves 2-26% and 13-22% better performance than CONGA and Hermes, respectively. After many experiments, we observe a similar trend for DCTCP in section IV, except that CONGA and other flowlet-based schemes performs better. This is because TCP is more bursty, which creates more sufficient flowlet gaps.



(a) Web-search (b) Data-mining

Fig. 8: Overall avg FCT in the asymmetric topology (normalized to IntFlow).

V. Discussion

Effectiveness of the rerouting design: Compared with solutions based on per-packet switching, IntFlow obtains more good rerouting opportunities by exploring traffic characteristics with flowlet switching. And compared to pure flowlet switching, IntFlow achieves more faster reaction to congestion, especially when traffic is too smooth to create enough new flowlets. Besides, IntFlow pushes congestion awareness to network devices for path selection in load balancing to achieve best visibility. Programmable switches give us the opportunity to implement this mechanism. As a result, IntFlow achieves the best performance in almost all cases in our evaluation.

Parameter settings and processing overhead in the network: IntFlow introduces several parameters to sense network failures/congestion to make proactive load balancing decisions. In this paper, we provide some tuned values for parameter settings according to previous work and our tests. But an automatic parameter tuning procedure for the optimal parameter settings should greatly simplify the implement of IntFlow. We consider it as a future work. IntFlow requires per flow forwarding state implemented by an flowlet table, which can be feasible at low cost in leaf switches for concurrent flowlets[12]. Recent hardware switch architectures make it feasible to perform flexible packet processing inside the network[26]. This allows some load balancing schemes to be implemented in the network, such as CONGA[12], HULA[13], MP-HULA[37]. IntFlow similarly calculates and relays congestion information in the network.

VI. Related Work

We briefly discuss related work that has informed and inspired our design.

Some centralized mechanisms (e.g. Hedera[2], MicroTE[6] and FastPass[38]) employ centralized schedulers to monitor global network state and schedules large flows in multiple paths. But they have long scheduling intervals. It is not adaptive to the traffic volatility of datacenter networks. These centralized mechanisms cannot meet the requirements of delay-sensitive applications in datacenters.

Some in-network solutions (e.g. RPS[8], DRILL[10], LetFlow[24], CONGA[12], HULA[13] and AG[11]) employ custom switches to balance traffic. They spray the fixed switching units, including per-packet and per-flowlet, across available paths to make full use of link resource. They explore traffic bursts to optimize load balancing performance. But this kind of schemes still have some problems under the unpredictable network environment in datacenters. RPS and DRILL are prone to experience packet reordering under asymmetric topology[5], [11]. And they also suffers from congestion mismatch under asymmetry. Besides, performance optimization effect of flowlet-based schemes may be less noticeable under smoother workloads. AG adjusts the switching granularity according to the latency-based congestion conditions of all paths to adapt to different degrees of topology asymmetry. But it cannot balance load based on flow status in the network, which may cause performance issues under failed switches, such as silent random packet drops and packet blackholes[23], [5]. Therefore, though in-network schemes are likely to provide sufficient rerouting opportunities, it is difficult for them to achieve adaptive load balancing according to flow status.

Many schemes performs load balancing at end hosts. Presto[9] routes flowcells to balance load at network edge. But Presto performs poorly with static weights under asymmetry and suffers from congestion mismatch. CLOVE-ECN[14] leverages per-flowlet weighted round robin at end hosts to route flowlets, which provides ease of deployment but achieves worse performance compared to flowlet-based schemes implemented in switches. These path weights are calculated according to ECN signals residing in ACKs. MPTCP[36] designed as a transport protocol routes several subflows simultaneously over multiple paths. MPTCP creates more burstiness and performs poorly under incast[12]. And these subflows still use ECMP to distribute traffic, which can cause ECMP hash collisions. Flowbender[16] reroutes flows blindly when congestion is detected based on ECN signals of every flow at end hosts. Without global awareness of congestion and with a random load balancing decision Flowbender has sub-optimal performance. Hermes[5] uses packet as the minimum switchable granularity, which exploits ECN signals and coarse-grained RTT measurements to sense congestion on multiple paths. Though schemes rerouting flows at packet granularity based on flow status can make proactive and cautious load balancing decisions, they lose the rerouting opportunity based on traffic bursts.

Finally, all the aforementioned schemes try to solve problems from one dimension. They either passively wait for new flowlets to rerouting flows or proactively reroute flows at packet granularity based on flow status. IntFlow achieves proactive and cautious load balancing by integrating per-packet and per-flowlet switching strategy to improve performance.

VII. Conclusion

We propose IntFlow, a novel load balancing solution, which combines the advantages of monitoring flow status at end hosts and in-network congestion-aware flowlet switching. IntFlow assesses flow status at end hosts to assist the flowlet switching at programmable switches. IntFlow reacts to congestion timely based on flow status to achieve proactive rerouting at end hosts, while performing cautious rerouting for flowlet switching in the network. We evaluate IntFlow through large-scale simulations. Experimental results show considerable performance improvements compared to current schemes.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceeding of the ACM SIGCOMM*, 2008, pp. 63–74.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceeding of the USENIX NSDI*, 2010, pp. 89–92.

[3] A. Greenberg *et al.*, "VL2: a scalable and flexible data center network," in *Proceeding of the ACM SIGCOMM*, 2009, pp. 51–62.

[4] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC 2992*, 2000.

[5] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proceeding of the ACM SIGCOMM*, 2017, pp. 253–266.

[6] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proceeding of the ACM CoNEXT*, 2011, p. 8.

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

[8] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proceeding of the IEEE INFOCOM*, 2013, pp. 2130–2138.

[9] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," in *Proceeding of the ACM SIGCOMM*, 2015, pp. 465–478.

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

[11] J. Liu, J. Huang, W. Li, and J. Wang, "AG: Adaptive switching granularity for load balancing with asymmetric topology in data center network," in *Proceeding of the IEEE ICNP*, 2019, pp. 1–11.

[12] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proceeding of the ACM SIGCOMM*, 2014, pp. 503–514.

[13] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: Scalable load balancing using programmable data planes," in *Proceeding of the ACM SOSR*, 2016, p. 10.

[14] N. Katta *et al.*, "Clove: Congestion-aware load balancing at the virtual edge," in *Proceeding of the ACM CoNEXT*, 2017, pp. 323–335.

[15] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCP's burstiness with flowlet switching," in *Proceeding of the ACM HotNets*, 2004.

[16] A. Kabbani, B. Vamanan, J. Hasan, and F. Duchene, "FlowBender: Flow-level adaptive routing for improved latency and throughput in datacenter networks," in *Proceeding of the ACM CoNEXT*, 2014, pp. 149–160.

[17] "Barefoot Tofino," Accessed June 25, 2019. [Online]. Available: https://barefootnetworks.com/.

[18] "Intel ethernet switch fm6000 series, white paper," 2013.

[19] S. Guenender *et al.*, "NoEncap: overlay network virtualization with no encapsulation overheads," in *Proceeding of the ACM SOSR*, 2015, p. 9.

[20] "NS3," Accessed June 25, 2019. [Online]. Available: https://www.nsnam.org/.

[21] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceeding of the ACM IMC*, 2010, pp. 267–280.

[22] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 350–361, 2011.

[23] C. Guo *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceeding of the ACM SIGCOMM*, 2015, pp. 139–152.

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

[25] M. Alizadeh *et al.*, "Data center tcp (dctcp)," in *Proceeding of the ACM SIGCOMM*, 2010, pp. 63–74.

[26] N. K. Sharma *et al.*, "Evaluating the power of flexible packet processing for network resource allocation," in *Proceeding of the USENIX NSDI*, 2017, pp. 67–82.

[27] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, p. 87–95, 2014.

[28] J. Li, E. Michael, and D. R. K. Ports, "Eris: Coordination-free consistent transactions using in-network concurrency control," in *Proceeding of the ACM SOSP*, 2017, pp. 104–120.

[29] X. Jin *et al.*, "NetCache: Balancing key-value stores with fast in-network caching," in *Proceeding of the ACM SOSP*, 2017, pp. 121–136.

[30] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica, "DistCache: provable load balancing for large-scale storage systems with distributed caching," in *Proceeding of the USENIX FAST*, 2019, pp. 143–157.

[31] W. Bai *et al.*, "Information-agnostic flow scheduling for commodity data centers." in *Proceeding of the USENIX NSDI*, 2015, pp. 455–468.

[32] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "DX: Latency-based congestion control for datacenters," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 1, pp. 335–348, 2017.

[33] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," in *Proceeding of the ACM SIGCOMM*, 2015, pp. 537–550.

[34] Q. Shi, F. Wang, D. Feng, and W. Xie, "ALB: Adaptive load balancing based on accurate congestion feedback for asymmetric topologies," in *Proceeding of the IEEE IWQoS*, 2018, pp. 1–6.

[35] C. Kim *et al.*, "In-band network telemetry via programmable dataplanes," in *Proceeding of the ACM SIGCOMM demo paper*, 2015, pp. 42–44.

[36] C. Raiciu *et al.*, "Improving datacenter performance and robustness with multipath TCP," in *Proceeding of the ACM SIGCOMM*, 2011, pp. 266–277.

[37] C. H. Benet, A. J. Kassler, T. Benson, and G. Pongracz, "MP-HULA: Multipath transport aware load balancing using programmable data planes," in *Proceeding of the ACM SIGCOMM Workshop NetCompute*, 2018, pp. 7–13.

[38] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized "zero-queue" datacenter network," in *Proceeding of the ACM SIGCOMM*, 2014, pp. 307–318.

Qingyu Shi He received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2014. He is currently a PhD student majoring in Computer Architecture in Wuhan National Laboratory for Optoelectronics (WNLO). His current research interests include software-defined networking, datacenter networks and distributed storage systems. He has several publications in major journals and international conferences, including CN and IWQoS.

Fang Wang She received her BE degree and Master degree in computer science in 1994, 1997, and Ph.D. degree in computer architecture in 2001 from Huazhong University of Science and Technology (HUST), China. She is a professor of computer science and engineering at HUST. Her interests include distribute file systems, parallel I/O storage systems and graph processing systems. She has more than 50 publications in major journals and international conferences, including FGCS, ACM TACO, SCIENCE CHINA Information Sciences, Chinese Journal of Computers and HiPC, ICDCS, HPDC, ICPP.

Dan Feng She received the BE, ME, and PhD degrees in Computer Science and Technology in 1991, 1994, and 1997, respectively, from Huazhong University of Science and Technology (HUST), China. She is a professor and vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 100 publications in major journals and international conferences, including IEEE-TC, IEEE-TPDS, ACM-TOS, JCST, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP. She serves on the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012. She is a member of IEEE and a member of ACM.