Harnessing TCP's Burstiness with Flowlet Switching

Shan Sinha ssinha@mit.edu
Srikanth Kandula kandula@mit.edu
Dina Katabi dk@mit.edu

Abstract— **TCP's burstiness is usually regarded as harmful, or at best, inconvenient. This paper adopts a new perspective and examines whether TCP's burstiness is useful for certain applications. It shows that the burstiness can be harnessed to make TCP more robust to packet reordering caused by route change. We define a flowlet as a burst of packets from the same flow followed by an idle interval. We develop a scheme that uses flowlets to split traffic across multiple parallel paths. We show that flowlet switching is an ideal technique for load balancing traffic across multiple paths as it has the accuracy of packet switching, combined with the robustness of flow switching to packet reordering. The accuracy, simplicity, and low-overhead of flowlet switching makes it a strong candidate for replacing the current hash-based schemes used in routers for splitting traffic across multiple links. In particular, when the desired split ratios vary over time, flowlet switching accurately splits traffic across multiple paths whereas current hash-based schemes are highly inaccurate. Hence flowlet switching provides a key component for research in the areas of realtime adaptive multipath routing and fine-grained traffic engineering.**

I. Introduction

Splitting traffic across multiple paths/links according to some desired ratios is an important functionality for network management. Many commercial router vendors, such as Cisco and Juniper, provide basic support for it in their products [10, 16]. It is also a key enabling technology for much research in the areas of traffic engineering [7, 27] and adaptive multipath routing [17, 28], which balances the load across multiple paths to reduce congestion and increase availability. Another potential application for traffic splitting includes adaptive multihoming, which allows a stub domain to adaptively split its traffic across multiple access links connected to different ISPs to optimize performance and cost [2, 14].

Traffic splitting is a challenging problem because of the tradeoff between achieving low deviation from the desired traffic ratios (i.e. high accuracy) and avoiding packet reordering, which hinders TCP performance. Two main approaches exist for splitting traffic. The first is packet-based splitting, which assigns each packet to a path with a probability proportional to the path's desired traffic share and independent of the assignment of other packets [7, 20]. This method ensures the resulting allocation accurately matches the desired split ratios but may allocate packets from the same flow to different paths, causing reordering and confusing TCP congestion control. Some proposals aim to make TCP less vulnerable to reordered packets [6, 19, 31], which, if widely deployed, would make packet-based splitting more robust. But prior experiences suggest such wide-scale deployment is unlikely in the near future.

Instead, routers [10, 16] use variations of flow-based splitting, assigning all packets of a flow to a single path. In contrast to packet-based splitting, this approach avoids reordering but cannot accurately achieve the desired splitting ratios [24]. Distributing traffic in units of flows rather than packets reduces the resolu-

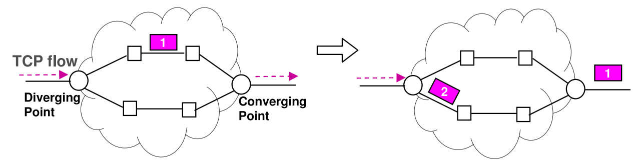

Fig. 1. **If the first packet leaves the convergence point before the second packet reaches the divergence point, one can assign the second packet to a new path without risking TCP packet reordering.**

tion of the splitting scheme. Further, flows differ greatly in their rates [22, 25, 32]. Assigning an entirely new TCP flow to a path makes the change to the path's rate unpredictable. Prior work tried to estimate the rate of each flow and used these estimates when mapping flows to paths, but found these rates to be unstable and change quickly during the lifetime of a flow [22]. The inaccurate traffic splitting resulting from pinning a flow to particular path leads to an unbalanced load and potentially worse performance. It may also lead to extra cost if the domain is charged differently for sending traffic on different parallel links, as in adaptive multihoming [14].

Ideally, one would like to combine the accuracy and low-overhead of packet-based splitting with the ability of flow-based splitting to avoid reordering TCP packets. This paper shows that *flowlet-based splitting* achieves the best of both these worlds.

A flowlet is a burst of packets from a given TCP flow. Flowlets are characterized by a timeout value, δ, which is the minimum inter-flowlet spacing, i.e., packet spacing within a flowlet is smaller than δ.

Flowlet-based splitting exploits a simple observation. Consider a set of parallel paths, which diverge at a particular point and converge later, each containing some number of hops. Given two consecutive packets in a TCP flow, if the first packet leaves the convergence point before the second packet reaches the divergence point, one can route the second packet —and subsequent packets from this flow— on to any available path with no threat of reordering, as in Fig. 1. *Thus, by picking flowlet timeout larger than the maximum latency of the set of parallel paths, consecutive flowlets can be switched independently with no danger of packet reordering.* In fact, for any set of parallel paths, we can further tighten the timeout value to the difference between the maximum and minimum path latencies. We call this maximum delay difference, the Minimum Time Before Switch-ability (MTBS). As long as the flowlet timeout, δ, is larger than the MTBS, flowlet switching does not cause packet reordering.

FLARE: We have developed FLARE, a flowlet aware routing engine, which uses flowlet switching to split traffic across multiple paths or links according to some desired split ratios.

Analyzed on traces collected at a major peering point, a stub domain border router, and various backbone routers, FLARE's errors (i.e., deviation from desired splits) are often as low as packet-based splitting and an order of magnitude lower than flow-based schemes, including the hash-based approach currently used by routers. Using FLARE is particularly advantageous when the desired splitting ratios vary with time, in which case the errors resulting from pinning every flow to a single path may reach 65% (see §IV).

FLARE is also tolerant to misconfiguration. The probability of reordering, for misconfigured values of δ, changes slowly with its deviation from MTBS. For example on the peering trace, a deviation of up to 100 ms produces, at maximum, a 0.06% probability of triggering 3 dup-acks. This number is negligible in comparison with typical drop probabilities in the Internet [3, 23], and thus on average is unlikely to affect TCP's performance.

Finally, FLARE's overhead is limited to maintaining a small table of approximately 1000 entries and calculating a single hash per packet (which is what routers currently do when splitting traffic [10, 16]). The low errors, simplicity, and minimal overhead of FLARE make it a candidate for replacing current traffic splitting mechanisms in the routers.

Harnessing TCP's burstiness with flowlets: Prior work either characterizes TCP's burstiness [12, 15, 30, 33], or proposes mechanisms for smoothing it [1]. We adopt a new perspective and explore whether TCP's burstiness is useful for certain applications. We describe the use of flowlets as a way to harness TCP's burstiness to improve the performance of traffic splitting across multiple paths. This paper reveals intrinsic properties of flowlets. Our results show that contrary to intuition, the origins of flowlets are not limited to very short flows, flows that are starting with small windows (1-2 pkts), or flows suffering from timeouts. The major source of flowlets is the burstiness of TCP at RTT and sub-RTT scales. As a result, most flowlets are either a whole or a fraction of a congestion window. Further, though the number of flowlets is usually an order of magnitude larger than the number of flows in a trace, the number of concurrent flowlets, at any point in time, is two orders of magnitude smaller than the number of concurrent flows. As a result, a small hash table with a few hundred entries is enough to track flowlets.

This paper is a promising first step. Packet reordering is the first hurdle to overcome when spreading a single flow across multiple paths. We plan to study the interaction between multipath routing and TCP's RTT estimators and its congestion window adaptation algorithm. While some prior work on these topics exists, the problems are far from being solved. If successful, our work will enable adaptive multipath routing and adaptive multihoming to deliver on their promises of better performance and increased availability.

II. FLARE: Flowlet Aware Routing Engine

A. The Splitting Problem

The traffic splitting problem is formalized as follows [24]. The aggregate traffic arriving at a router, at rate R, is composed of a number of distinct transport-layer flows of varying rates. Given N disjoint paths, which can be used concurrently, and a split vector $\vec{F} = (F_1, F_2, ..., F_N), F_i \in [0,1]$ and $\sum_{i=1}^{i=N} F_i = 1$, split the aggregate traffic into N portions such that the traffic rate flowing on path i is equal to $F_i \times R$.

The splitting problem is a key component of the general problem of load balancing. In addition to a traffic splitter, balancing the load across multiple paths requires a mechanism to find the splitting vector $\vec{F}$. Depending on the environment, the network administrator may set $\vec{F}$ to a static value, or use an adaptive routing protocol to dynamically adapt $\vec{F}$ to the state of the network.

B. Design & Implementation

FLARE accurately splits traffic across multiple paths, while minimizing TCP packet reordering. FLARE resides on a router that feeds multiple parallel paths and takes as input a split vector, that could change over time. Upon receiving a packet, FLARE determines the best path along which to route the packet to achieve the desired split vector, and forwards the packet appropriately. [1]

FLARE relies on the flowlet abstraction to accurately split TCP traffic along multiple paths without causing reordering. The network administrator configures FLARE with a flowlet timeout value δ. The administrator uses knowledge of the network to pick a δ larger than the MTBS, the maximum delay difference between the set of parallel routes under consideration. This choice of δ lets FLARE assign flowlets of the same flow to different parallel paths, without causing TCP packet reordering. As we show in §IV, $\delta \in [50, 100]ms$ produces good accuracy in general. Further, misconfiguring δ by up to 100ms causes negligible reordering.

Packets for which transport-layer performance is unaffected by reordering may be allocated to any path. For simplicity, we refer to these packets as non-TCP packets. Since routing flowlets will typically be slightly less accurate than a packet-based splitter, FLARE uses non-TCP packets to balance residual error occurring from routing flowlets.

FLARE is configured with a flowlet timeout δ and has two components: a token-counting algorithm and a flowlet assignment algorithm.

Token-counting algorithm: FLARE assigns a token counter, t_i to each path i of the set of parallel paths. For every packet of size b bytes, all token counters are updated as follows:

$$t_i = t_i + F_i \times b, \qquad \forall i$$

where F_i is the fraction of the load to be sent on path i. If the packet is a non-TCP packet, it is assigned to the path with the maximum number of tokens. Otherwise, it is assigned according to the flowlet-to-path assignment algorithm. In either case, once the packet has been assigned to a particular path j, the corresponding token counter is decremented by the size of the packet:

$$t_j = t_j - b.$$

Flowlet-to-path assignment: FLARE uses a hash table that maps flowlets to paths. Each table entry contains two fields `last_seen_time` and `path_id`. When a packet arrives, FLARE computes a hash of the source IP, destination IP, source port and destination port.[2] This hash is used as the key into the flowlet table. If the current time is smaller than `last_seen_time + δ`, the packet is sent on the path identified

[1] FLARE actually hands the packet to the next stage toward transmission on the appropriate link (e.g., an output queue).

[2] The authors of [8] recommend a CRC-16 hash.

Trace	Date	Duration	Packets	# Flows	Avg. Flow Rate	Max. Flow Rate	% Bytes Non-TCP
Peering	Mar 5, 2003, 7 PM	12 minutes	9.15 million	454K	1.83 Kbps	6.01 Mbps	7.97%
LCSout	May 9, 2003, 6 PM	1 hour	25.4 million	426K	13.73 Kbps	75.29 Mbps	2.80%
NLANR-1	Mar 7, 2004, 3 AM	90 seconds	7.3 million	340.5K	7.74 Kbps	50.89 Mbps	13.1%
NLANR-2	Apr 15, 2003, 8 PM	90 seconds	1.69 million	10K	31 Kbps	98.1 Mbps	12.1%

TABLE I. **Datasets used in evaluation.**

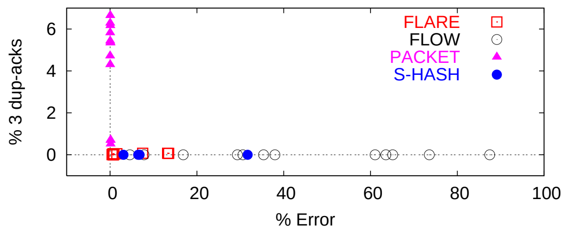

Fig. 2. **Visualization of results in Table II. Points near the origin represent performance with low error and low reordering. FLARE's performance falls within this region. The performance of S-HASH is close, but S-HASH cannot be used when the split vector is dynamic.**

Trace	Static			
	FLARE	PACKET	FLOW	S-HASH
Peering	0.47% (0.01%)	0.12% (0.71%)	4.54% (0%)	6.47% (0%)
LCSout	7.50% (0.07%)	0.07% (6.17%)	37.98% (0%)	31.69% (0%)
NLANR-1	0.55% (0.02%)	0.02% (4.72%)	30.60% (0%)	6.79% (0%)
NLANR-2	0.70% (0.05%)	0.03% (6.29%)	35.36% (0%)	3.08% (0%)
Trace	**Mildly Dynamic**			
	FLARE	PACKET	FLOW	S-HASH
Peering	0.75% (0.00%)	0.11% (0.69%)	7.82% (0%)	–
LCSout	13.48% (0.07%)	0.09% (5.34%)	65.10% (0%)	–
NLANR-1	0.82% (0.01%)	0.03% (5.43%)	16.86% (0%)	–
NLANR-2	0.78% (0.04%)	0.05% (6.65%)	73.55% (0%)	–
Trace	**Dynamic**			
	FLARE	PACKET	FLOW	S-HASH
Peering	0.74% (0.00%)	0.12% (0.52%)	29.33% (0%)	–
LCSout	13.26% (0.07%)	0.09% (5.36%)	63.54% (0%)	–
NLANR-1	0.90% (0.02%)	0.02% (4.30%)	61.05% (0%)	–
NLANR-2	1.56% (0.05%)	0.03% (5.83%)	87.47% (0%)	–

TABLE II. **FLARE's accuracy is an order of magnitude higher than flow-based and static-hash splitting schemes and its robustness to reordering is an order of magnitude higher than packet-based splitting. Values outside the parenthesis are errors; numbers inside the parenthesis are the probability of mistakenly triggering 3 dup acks. Note that this experiment utilized the values, δ=60 ms, MTBS=80 ms. FLARE's reordering arise from this slight misconfiguration of δ. When δ=MTBS, FLARE shows no reordering.**

by `path_id` and `last_seen_time` is set to current time. Otherwise, the packet begins a new flowlet and may be assigned to a new path. FLARE assigns new flowlets to the path with the maximum number of tokens, sets `path_id` to the new path id, and sets `last_seen_time` to current time.

III. Experimental Environment

Packet Traces: We use traffic traces from four sources. First, the Peering trace is collected at multiple 622 Mbps peering links from the same router connecting a Tier-1 ISP to two large ISPs. Second, the LCSout trace is collected at the border router connecting MIT's Computer Science and Artificial Intelligence Lab to the Internet over a 100 Mbps link. Finally, NLANR-1 and NLANR-2 are sets of backbone traces collected by NLANR on OC12 and OC3 links [21], respectively. Table I summarizes relevant information about these traces (flow rates are computed according to [32]). In all traces, TCP constitutes over 85% of the traffic, with the LCSout trace being 97% TCP.

Methodology: We imagine the router at which the trace is collected to be feeding multiple parallel paths, and splitting the traffic among them according to a desired split vector. We evaluate how well FLARE tracks the desired splits while avoiding TCP reordering. The experiments depend on these parameters:

- $\vec{F}$, the split vector, specifying the fractions at which incoming traffic needs to be split. In our experiments, we use both a static vector $\vec{F_s} = (.3, .3, .4)$, and a dynamic vector

$$\vec{F_d}(t) = .13(1, 1, 1) + .6(sin^4 x, sin^2 x \cdot cos^2 x, cos^2 x),$$

  where $x(t) = \frac{2\pi t}{p}$. We use two dynamic vectors, $\vec{F}_{d1}$, which reflects changes over long time scales (p=40min), and $\vec{F}_{d2}$, which reflects changes over short time scales (p=4min). The amplitude and period of $\vec{F_d}$ are chosen based on [2]. The period of the fast changing vector is chosen based on [9], which states that congestion spikes lasting for 5 to 10 minutes occur in ISP networks.
- δ, the flowlet timeout interval. Unless specified otherwise, $\delta = 60$ms. This means that we are simulating a situation in which the administrator thinks that the delay difference between the various parallel paths is less than 60 ms. Given current values for one-way delay in the Internet (e.g., coast-to-coast is typically < 40ms), a delay difference of 60 ms or less is applicable to many possible cases of parallel paths.
- MTBS, the actual maximum delay difference between the parallel paths. Unless specified otherwise, MTBS=80 ms. By making MTBS different from δ, we simulate errors in the administrator's estimate of MTBS.
- T_{avg}, the time window over which the paths' rates are computed to measure whether they match the desired split. This is a measurement parameter irrelevant to the operation of FLARE. We fix $T_{avg} = 0.3s$.[3]
- S_{hash}, the size of hash table used by FLARE. Unless otherwise specified, we set $S_{hash} = 2^{10}$ entries.

Measuring Accuracy: An optimal traffic splitting policy ensures that path i receives a fraction of the traffic F_i on any timescale, but the actual fraction of traffic sent on i is F'_i. We measure the splitting error as:

$$Error = \frac{1}{N} \sum_{i=1}^{N} \frac{|F_i - F'_i|}{F_i}, \quad (1)$$

where N is the number of parallel paths among which the traffic is divided. The graphs report the average error over non-overlapping windows of size T_{avg}. Accuracy is $1 - Error$.

[3]The exact value of this parameter is not important as long as it is small enough to show the instantaneous variability of the load. We chose $T_{avg} = 0.3s$ because this is the update interval of TeXCP [17], an adaptive multipath routing protocol. Also, routers can typically buffer about 250ms worth of data [5].

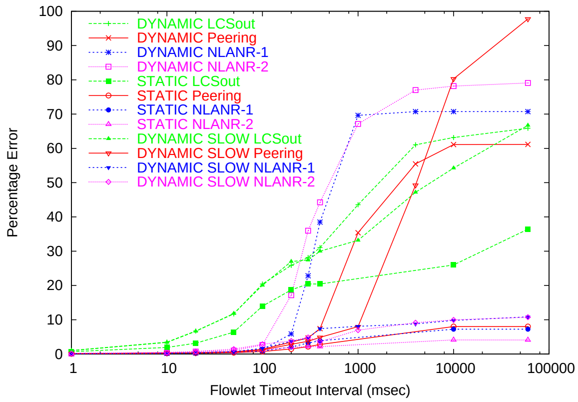

Fig. 3. **A flowlet timeout in the range $[50, 100]$ms produces good accuracy. Errors as a function of flowlet timeout interval δ for the static split, $\vec{F}_s$, the mildly dynamic split, $\vec{F}_{d1}$, and the dynamic split $\vec{F}_{d2}$.**

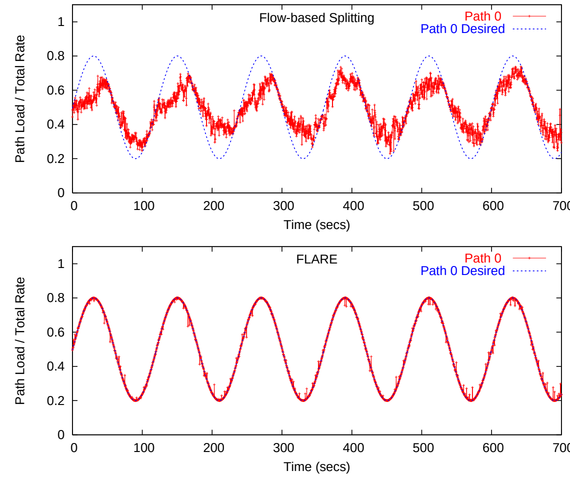

Fig. 4. **In contrast to flow-based splitting, FLARE is suitable for adaptive multipath routing protocol as it can accurately track a varying split vector. Graphs are for the peering trace, 60ms flowlet timeout, and 2 paths with a sinusoidal splitting function. For clarity, we show the load on one path.**

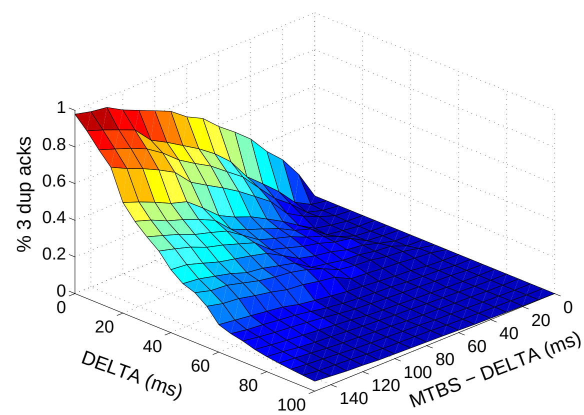

Fig. 5. **Percentage of packets that lead to three duplicate acks vs. flowlet timeout interval (DELTA δ) and the MTBS of a network, for the peering trace. Packet-based splitting causes up to 1% of the packets to trigger TCP window reduction, which is about the loss rate in the Internet. FLARE with $\delta = 60ms$ causes fewer than .06%, even when the actual MTBS is larger than δ by 100ms.**

Measuring TCP Disturbance: We estimate the disturbance of a certain splitting algorithm as the probability that a packet triggers 3 dup-acks due to the reordering caused by the splitting scheme.

Simulating Splitting Schemes: We use Deficit Round Robin to simulate packet-based traffic splitting [26], which is highly accurate [24]. We simulate flow-based splitting by assigning each new flow to the path farthest away from its desired traffic share, and retaining the assignment for the duration of the flow. We also simulate a static-hash splitting (S-HASH) scheme by hashing the arriving packet's source and destination IP addresses and ports into a large space, then allocating the hash space to the various paths proportionally to their desired traffic shares [24].

IV. Performance of Flowlet-Based Splitting

Comparison with Other Splitting Schemes Table II and Fig. 2 compare FLARE with other splitting schemes vis-a-vis accuracy and TCP disturbance. The results in the table are computed using δ=60ms and MTBS=80ms, which is a slight misconfiguration of δ. In our experiments, FLARE provides a good tradeoff between accuracy and robustness to reordering. Its errors are an order of magnitude lower than flow-based and static hash splitting schemes, and its tendency to trigger TCP congestion window reduction is an order of magnitude less than that of packet-based splitting. The table also shows that packet-based splitting is inadequate for these scenarios because it triggers 3 dup ack events at a rate comparable to or higher than the loss rate in the Internet [4, 23]. Finally, the S-HASH scheme, though 10 times less accurate than FLARE, has a reasonable splitting accuracy for a static split vector, but does not react to dynamic split vectors.

Accuracy of Flowlet Splitting First, we show that flowlet-based splitting is accurate for realistic values of δ. Fig 3 shows the error as a function of flowlet timeout for our four traces. The figure shows results for the split vectors: $\vec{F}_s$, $\vec{F}_{d1}$, and $\vec{F}_{d2}$. The figure shows that on all traces other than the LCSout trace, flowlet-based splitting achieves an accuracy comparable to packet-based splitting, as long as δ <100 ms. Given typical values for one-way delay in the Internet (e.g., coast-to-coast delay is less than 40ms), a delay difference in the range $[50, 100]$ms should apply to many possible sets of parallel paths.

The errors on the LCSout trace are higher. For this domain, the administrator might want to pick δ=60ms, which results in an error of 7%-14% depending on how quickly the split vector changes. Despite the relatively high error, other schemes that do not reorder packets have substantially more errors on the LCSout trace (see Table II). We attribute the higher errors in the LCSout trace to two factors. This trace contains a large amount of local intra-MIT traffic with a small RTT. Also, it has a low fraction of non-TCP traffic (<3% of the bytes), which prevents FLARE from compensating for residual errors. We note that FLARE does not require non-TCP traffic to perform well because FLARE performs well on the other 3 traces even when all non-TCP traffic is removed.

Second, Fig. 4 compares how flowlet- and flow-based splitting track a changing split vector in real-time, a feature required by adaptive multipath routing [11, 17]. The figure represents δ=60 ms and two paths with a split that varies along a sinusoidal wave with a period of 2 minutes ($\vec{F} = 0.2(1, 1) + 0.6(sin^2 x, cos^2 x)$). In this experiment, FLARE tracks the desired split much more closely than the flow-based splitter.

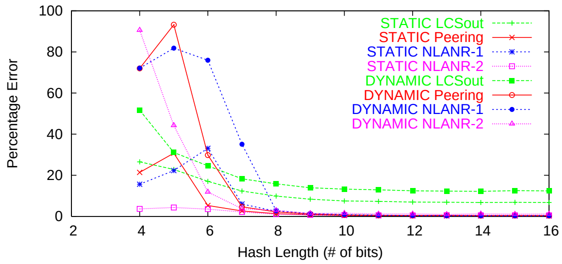

Fig. 6. **Error as a function of the flowlet table size for both static $\vec{F}_s$ and dynamic $\vec{F}_{d2}$ split vectors. FLARE achieves low errors without storing much state. A table of 2^{10} five-byte entries is enough for the studied traces.**

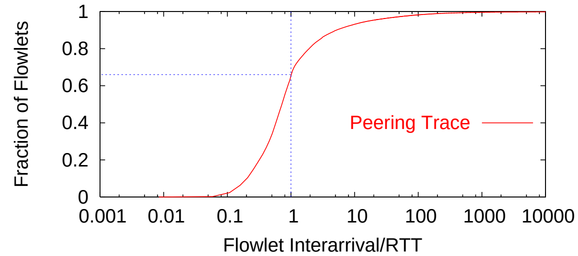

Fig. 7. **CDF of flowlet interarrival time normalized by flowlet RTT. About 68% of the 60ms-flowlets have sub-RTT interarrivals, indicating that most of these flowlets are a whole or a portion of a congestion window.**

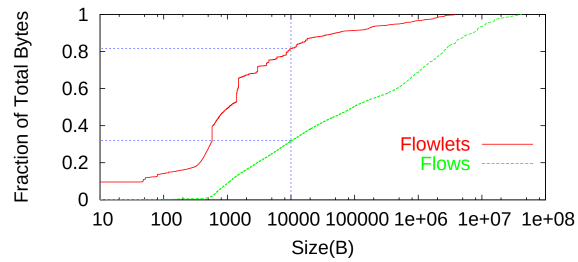

Fig. 8. **More than 70% of bytes are in 60ms-flowlets of size smaller than 2KB. This indicates that the concept of flowlets shift most of the bytes to small flowlets, which can be independently switched.**

Trace	Arrival Rate (/sec)		#Concurrent	
	Flows	Flowlets	Flows	Flowlets
LCSout	143.16	1454.98	1450.42 (2030)	18.41 (49)
Peering	611.95	8661.43	8477.33 (8959)	28.08 (56)
NLANR-1	3784.10	35287.04	47883.33 (57860)	240.12 (309)
NLANR-2	111.33	2848.76	1559.33 (1796)	50.66 (71)

TABLE III. **60ms-Flowlets arrive at a much higher rate than flows; but there are much fewer concurrent flowlets than flows. The values outside parenthesis are averages while the numbers inside are the maximum values.**

TCP's Disturbance We also evaluated FLARE's sensitivity to flowlet timeout values smaller than the actual MTBS. Such a choice of δ will result in TCP packet reordering. Fig. 5 shows the probability of mistakenly triggering 3 dup acks, as a function of δ and $MTBS - \delta$, for the case of 3 paths with a static split vector $\vec{F}_s$, path latencies $(x, x + 0.5\ MTBS, x + MTBS)$, on the Peering trace.

The figure shows that FLARE is tolerant to bad choices of the flowlet timeout, i.e., choices in which δ is smaller than the actual MTBS. In particular, we have seen that, choosing δ in the range $[50, 100]$ms achieves good accuracy on our traces. Fig. 5 shows that for any $\delta > 50ms$, the percentage of packets that trigger a TCP window reduction is less than 0.06%, even when the actual MTBS is larger than the chosen δ by 100 ms. This number is negligible in comparison with typical drop probabilities in the Internet [3, 23], and thus on average is unlikely to impact TCP's performance. In general, for $\delta > 50$ms, the probability of 3 dup ack occurrences increases slowly as the difference between MTBS and δ increases. In other words, a misconfigured FLARE, using a flowlet timeout smaller than the actual MTBS, may continue to perform reasonably well.

Overhead of Flowlet Splitting One of the most surprising results of this paper is the little overhead incurred by flowlet-based splitting. It requires edge routers to perform a single hash operation per packet and maintain a flowlet hash table, a few KB in size, which easily fits into the router's cache. We have estimated the required hash table size by plotting the splitting error, averaged over time windows $T_{avg} = 0.3s$, as a function of the hash length. For example, a hash length of 10 bits results in table of 2^{10} entries. Fig. 6 shows the error in our traces for both the static split vector $\vec{F}_s$ and the dynamic sinusoidal vector $\vec{F}_{d2}$. It reveals that the errors converge for a table size as small as 2^{10} entries.

V. Harnessing TCP's Burstiness

The idea underlying flowlet-based splitting is simple; instead of switching paths at the granularity of a packet or a flow, allow the router to switch bursts of packets from the same flow, as long as they are separated by a large-enough idle interval. Switching bursts of a few packets provides a higher resolution than flow-based switching, and thus can be more accurate. But a natural question to ask is why it is possible to divide most TCP flows into short flowlets, particularly the long ones, which contain the majority of total traffic [32]. Another is why tracking flowlets requires very little state even though the number of flowlets is larger than the number of flows. This section shows that by harnessing TCP's burstiness, flowlet-based splitting achieves an effectiveness that might appear puzzling at first.

Where Do Flowlets Come From? The origins of flowlets are not limited to very short flows, flows that are just starting with a small window of one or two packets, or flows that are suffering timeouts. These flowlet sources are not enough to make flowlet splitting as effective as §IV shows because most of the bytes are in the long TCP flows [32]. In fact, the main origin of flowlets is the burstiness of TCP at RTT and sub-RTT scales. Prior work has shown that the TCP sender tends to send a whole congestion window in one burst or a few clustered bursts and then wait idle for the rest of its RTT. This behavior is caused by ack compression, slow-start, and other factors [15, 30, 33]. This burstiness enables a FLARE router to consider a long TCP flow as a concatenation of short flowlets separated by idle periods, which is necessary for the success of flowlet-based splitting.

Figure 7 supports this argument. It was computed using the peering trace for δ=60 ms. The figure plots the time between arrivals of two consecutive flowlets from the same flow normalized by the RTT of the flow (RTT is computed using the MYSTERY TCP analyzer [18]). The graph shows that the vast majority of flowlets are separated by less than an RTT, indicating that a flowlet is usually a congestion window or a portion of it.

Why Flowlet Splitting is Accurate? Flowlet-based splitting is accurate due to two reasons. First, there are many more flowlets than flows, leading to many opportunities to rebalance an imbalanced load. Table III shows that flowlet arrival rates are an order of magnitude higher than flow arrival rates, in our traces. This means that in every second, flowlet-based splitting provides an order of magnitude more opportunities to rebalance an incorrect split than with flow-based splitting. Second, as shown in Fig. 8, most of the bytes are in small flowlets, allowing load rebalancing at a much higher granularity than at the size of a flow.

Why Flowlet Tracking Requires a Small Table? Despite the large number of flowlets in a trace, FLARE only needs to maintain state for currently active flowlets, i.e., flowlets that currently have packets in the network. Table III shows that the average number of concurrent flowlets is two orders of magnitude smaller than the number of concurrent flows. Indeed the maximum number of concurrent flowlets in our traces never exceeds a few hundreds. To track these flowlets without collision, the router needs a hash table containing approximately thousand entries, which is compatible with the results in §IV.

TCP's burstiness results in flow transmissions that take an on-off pattern– a burst of packets followed by an idle period [15]. This burstiness enables one to divide each long flow into multiple short flowlets. Much prior work advocates mechanisms to smooth TCP burstiness [1]. A paced TCP is useful for many applications. But since current TCP is bursty and is likely to stay bursty for the near future, it is beneficial to explore whether TCP burstiness can be useful. FLARE harnesses TCP burstiness to improve the performance of traffic splitting across multiple paths.

VI. Related Work

Traffic Splitting Mechanisms: Early work on traffic splitting considers forwarding packets onto multiple paths using some form of weighted round-robin or deficit round robin [26] scheduling. Others avoid packet reordering by consistently mapping packets to paths based on their endpoint information. Commercial routers [10, 16] implement the Equal-Cost Multipath (ECMP) feature of routing protocols such as OSPF and IS-IS. Hash-based versions of ECMP divide their hash space into equal-size partitions corresponding to the outbound paths, hash packets based on their endpoint information, and forward them onto the path whose boundaries envelop the packet's hash value [8, 28].

A few papers analyze the performance of various splitting schemes. Cao et al. evaluate the performance of a number of hashing functions on hash-based traffic splitting [8]. Rost and Balakrishnan [24] evaluate different traffic splitting policies, including rate-adaptive splitting methods.

Multipath Routing: The majority of proposed approaches to multipath routing require a method for traffic splitting across various parallel paths. Multipath routing sends traffic on multiple paths to balance the load and minimize the possibility of congestion [11, 13, 17, 28, 29]. Some of the work in this area focuses on adaptive approaches where the desired splitting vector varies with time and reacts to the observed load [11, 17]. This capability further constrains the splitting mechanism to be able to track a changing split vector, in addition to the standard requirements of achieving accuracy and maintaining packet order.

VII. Conclusion

To our knowledge, we are the first to introduce the concept of flowlet-switching and develop an algorithm which utilizes it. Our work reveals several interesting conclusions. First, highly accurate traffic splitting can be implemented with little to no impact on TCP packet reordering and with negligible state overhead. Next, flowlets can be used to make adaptive multipath routing more practical. Finally, the existence and usefulness of flowlets show that TCP burstiness is not necessarily a bad thing, and can in fact be used advantageously.

References

[1] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of TCP pacing. In *INFOCOM*, 2000.

[2] A. Akella, S. Seshan, and A. Shaikh. Multihoming performance benefits: An experimental evaluation of practical enterprise strategies. In *USENIX Tech. Conf.*, 2004.

[3] M. Allman, W. Eddy, and S. Ostermann. Estimating loss rates with tcp. *ACM Performance Evaluation Review*, 2003.

[4] D. Andersen, A. Snoeren, and H. Balakrishnan. Best-path vs. multi-path overlay routing. In *ACM IMC*, 2003.

[5] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *SIGCOMM*, 2004.

[6] E. Blanton and M. Allman. On making tcp more robust to packet reordering. In *ACM Computer Communication Review*, 2002.

[7] J. E. Burns et al. Path selection and b/w allocation in MPLS networks. *Perf. Eval.*, 2003.

[8] Z. Cao, Z. Wang, and E. W. Zegura. Performance of hashing-based schemes for internet load balancing. In *IEEE INFOCOM*, 2000.

[9] C. N. Chuah and C. Diot. A tier-1 isp prespective: Design principles & observations of routing behavior. In *PAM Workshop on Large-Scale Communications Networks*, 2002.

[10] Cisco express forwarding (cef). Cisco white paper, Cisco Systems., July 2002.

[11] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *IEEE INFOCOM*, 2001.

[12] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. Deriving traffic demands from operational ip networks: Methodology and experience. *ACM TON*, 2001.

[13] B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights in a changing world. In *IEEE INFOCOM*, 2000.

[14] D. K. Goldenberg, L. Qiu, H. Xie, Y. R. Yang, and Y. Zhang. Optimizing cost and performance for multihoming. In *ACM SIGCOMM*, 2004.

[15] H. Jiang and C. Dovrolis. The origin of tcp traffic burstiness in short time scales. Technical report, Georgia Tech., 2004.

[16] Junos 6.3 internet software routing protocols configuration guide. www.juniper.net/techpubs/software/junos/junos63/swconfig63-routing/html/.

[17] S. Kandula, A. Qureshi, S. Sinha, and D. Katabi. TeXCP: Intra-Domain Online Traffic Engineering with an XCP-Like Protocol. nms.lcs.mit.edu/ dina/texcp.

[18] S. Katti, C. Blake, D. Katabi, E. Kohler, and J. Strauss. M&M: Passive measurement tools for internet modeling. In *ACM IMC*, 2004.

[19] R. Ludwig and R. Katz. The eifel algorithm: Making TCP robust against spurious retransmissions. In *ACM Computer Communication Review*, 2000.

[20] D. Mitra and K. G. Ramakrishna. A case study of multiservice multipriority traffic engineering design. In *IEEE GLOBECOM*, 1999.

[21] National Laboratory for Applied Network Research. http://pma.nlanr.net/.

[22] K. Papagiannaki, N. Taft, and C. Diot. Impact of flow dynamics on traffic engineering design principles. In *IEEE INFOCOM*, Hong Kong, March 2004.

[23] V. Paxson. End-to-end internet packet dynamics. *ACM TON*, 1999.

[24] S. Rost and H. Balakrishanan. Rate-aware splitting of aggregate traffic. Technical report, MIT, 2003.

[25] M. Roughan et al. Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning. In *ACM IMW*, 2002.

[26] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM*, 1995.

[27] C. Villamizar. Mpls optimized multipath (mpls-omp), 1999. Internet Draft.

[28] C. Villamizar. Ospf optimized multi-path (ospf-omp), 1999. Internet Draft.

[29] Y. Wang and Z. Wang. Explicit routing algorithms for internet traffic engineering. In *IEEE ICCCN*, 1999.

[30] L. Zhang, S. Shenker, and D. D. Clark. Observations on the dynamics of a congestion control algorithm. In *SIGCOMM*, 1991.

[31] M. Zhang et al. RR-TCP: A reordering-robust tcp with dsack. In *IEEE ICNP*, 2003.

[32] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of internet flow rates. In *SIGCOMM*, 2002.

[33] Z.-L. Zhang, V. Ribeiro, S. Moon, and C. Diot. Small-time scaling behaviors of internet backbone traffic: An emprical study. In *INFOCOM*, 2003.