Improving TCP latency with super-packets

David Schinazi

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-80
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-80.html

May 15, 2014

Improving TCP latency with super-packets

by David Schinazi
david.schinazi@eecs.berkeley.edu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Committee:

Professor I. Stoica
Research Advisor

Professor S. Ratnasamy
Second Reader

Abstract

The vast majority of today's Internet traffic uses TCP connections whose lengths range from a few packets to many orders of magnitude more. Even though short flows are prevalent in web traffic and their completion time can directly impact user experience, these flows are generally not prioritized and can experience drops because of the load caused by longer TCP flows on routers. In this paper, we describe a novel way to improve the latency of these flows.

One possible solution to lower latency is to increase the Maximum Segment Size (MSS), which would allow transmitting as much information using fewer packets. However, this impairs efficient packet multiplexing by routers. Another approach to reduce latency is to increase TCP's initial window size, which also allows sending more data at once. However, the benefits of this for short flows can be wiped out if the flow experiences even a single packet drop as this will cause retransmissions and in some cases timeouts. In this paper, we propose the concept of super-packets to deal with this "all-or-nothing" property of flows. Super-packets are a collection of packets whose collective performance is critical for applications. End-hosts explicitly request reservation of buffer space for super-packets in routers who in turn look at their buffer utilization and guarantee forwarding of all packets in the given super-packet. This paper proposes a simple and efficient algorithm for such reservations in routers.

We use simulations to demonstrate how the use of super-packets has the same effect as increasing the MSS without the downsides. Extensive evaluation of an implementation on the Click Modular Router shows that super-packets can improve average latency of flows and also effectively solve the problem of incast. Additionally, our results demonstrate that super-packets can prioritize short flows without the need for explicit priorities.

1 Introduction

Many applications running over the Internet today rely on TCP. Most user requests or actions are conveyed through short TCP flows. Ensuring low latency for these flows is essential to a good user experience. However, in practice routers can be overloaded by traffic from longer flows and resort to dropping packets from short flows, significantly increasing latency. One way of optimizing latency is to modify the Maximum Transfer Unit (MTU), a fundamental artifact of packet switched networks. On the one hand a low MTU is good for packet multiplexing in routers since it enables a more efficient management and usage of memory. On the other hand, it limits the efficiency of the network, as many protocols directly depend on it. TCP is directly dependent on the MTU, as the Maximum Segment Size (MSS) must be set to be smaller than the MTU. This directly affects the achievable throughput, since it has long been known that the throughput of TCP is roughly proportional to the MSS (see Section 2). Throughput depends on the MSS mainly because drops occur at the granularity of TCP segments. Drops have a significant impact on the throughput, and by extension, on the end-to-end latency of flows [1].

In this paper, we propose a mechanism, called *super-packets*, by which we can decouple the throughput of TCP from the MSS and the MTU, allowing us to lower TCP latency while maintaining efficient packet processing. A super-packet is an abstract concept which consists of multiple IP packets that travel separately on the wire but are dropped at the granularity of a super-packet. Upon reception of the first packet in a super-packet, a router makes an admission decision based on its queue occupancy and then applies this decision to all subsequent packets in the same super-packet, ensuring that it will drop all of them or commit to attempting to deliver all of them. This leads to an *all-or-nothing* drop property with respect to super-packets.

Applying drop rates to entire super-packets has two advantages. First, throughput can be adjusted by changing the size of the super-packet, regardless of the underlying MTU. Second, latency-sensitive flows can benefit from the *all-or-nothing* drop property of super-packets, as many applications have a cliff effect with respect to drops. That is, the first dropped packet can often severely impact the latency of the application, but the subsequent drops do not impact the latency as much, until again a later packet again causes a cliff effect. This applies to the initial packets sent in a video stream, or live video/audio streaming. By having the size of a super-packet match the application-dependent performance characteristic, we can concentrate drops to entire super-packets, clearing queue space and

avoiding the cliff effect in many other flows. Put differently, rather than dropping few packets from many different latency-sensitive flows, we restrict drops to a few flows, which would anyway see a performance drop from their first dropped packet. Super-packets also allow us to protect vulnerable sequences of packets, such as small TCP windows or MPEG-1 frames.

In this paper, we demonstrate how super-packets can decouple MSS and MTU through the use of simulations. We also evaluate our implementation of super-packets using the Click Modular Router [2]. Our experiments show that super-packets improve the flow completion times in the case of incast (many flows converging on one bottleneck link). Interestingly enough, in some conditions even the flows that had their super-packets denied still perform better than the last flows when using regular packets. Also, in a very common case of short flows sharing the wire with long flows, super-packets can seriously reduce the completion time of short flows while barely affecting the bandwidth of long flows.

2 Background

The ever-increasing availability of always-on Internet connections has allowed the proliferation of applications running over the Internet. The usability of many of these applications is dependent on low latency, as higher latency makes for very bad user experience and can significantly impact revenue [3]. Today the main use of the Internet, measured in number of bytes transferred, is video streaming (e.g., YouTube, Netflix, Hulu) [4]. Most of these bytes are not latency-sensitive since these applications buffer data in advance. However, the start of these flows is latency-sensitive since the loading time of a video has been shown to have significant impact on user engagement [5].

In packet-switched networks such as IP, the Maximum Transfer Unit (MTU) determines how large an individual packet can be. Choosing a set value for MTU is a tradeoff. The higher the MTU, the more data can be sent at once with a fixed header length, increasing efficiency. However, a lower MTU allows better multiplexing to be achieved in routers, allowing both fast packet processing and interleaving of different flows. A larger MTU can also increase corruption rates and cause packets to be dropped when the checksum verification fails. For these reasons, the MTU across most of the IPv4 Internet today has been set to 1500 bytes. TCP relies on the notion of a Maximum Segment Size (MSS), which represents how much payload can be sent in one TCP packet or segment. It is equal to the MTU minus the header length.

Visualizing a network link as a pipe of given bandwidth, we could intuitively imagine that the throughput of a transport protocol only relies on the given capacity of these pipes and the drop rate of the transferred data. However, prior work using modeling, simulation and traffic observation has shown that the throughput of TCP is proportional to $\frac{\text{MSS}}{\text{RTT}\sqrt{p}}$, where p is the drop rate [6]. The appearance of the Round-Trip Time (RTT) in this equation is unexpected since one might imagine it influences latency but not throughput in the steady state. However, to avoid congestion, TCP increases its sending window until it detects congestion then backs off by halving this window. The RTT influences how often this window reduction can take place. Each drop will reduce throughput and all drops happen at the granularity of a MSS — each time a packet is dropped, MSS bytes of data must be retransmitted. Hence both the MSS and the drop rate appear in the equation. For this reason, the granularity of drops directly influences the throughput of a transport protocol like TCP. Our proposed mechanism, *super-packets*, changes the granularity of drops from individual packets to a super-packet, i.e., a collection of packets.

When considering short flows — i.e., TCP flows that span a very small number of TCP windows — the common metric is Flow Completion Time (FCT), represented by the time between the transmission of the first (SYN) packet and the reception of the last packet carrying useful data. One approach to reducing FCT is to increase the TCP initial congestion window size (CWND) to ten packets [7]. If we consider flows shorter than $10 \times$ MSS, we can hope for an optimal FCT of $2 \times$ RTT that can be achieved if no packets are dropped. However, a single drop from the initial CWND can cause a retransmission timeout (RTO), which is orders of magnitude greater than a RTT. Mechanisms such as TCP fast-retransmit and TCP Selective Acknowledgments have been designed to avoid this and reduce damage caused by a drop. However, if the dropped packet is one of the last ones in the flow, there may not be enough subsequent packets to trigger a triple duplicate acknowledgment [8]. Even if these mechanisms do trigger, the TCP CWND would still be halved and we would experience at least one RTT of delay in order to inform the sender of which packet has been dropped.

3 Super-Packets

A super-packet is a logical grouping of multiple packets. These packets travel separately on the wire but are considered to be part of a larger entity, a super-packet. Routers treat super-packets differently: instead of dropping data at the

granularity of a packet, routers either drop all packets in a super-packet or commit to forwarding all of them. End hosts mark packets they send as super-packets to indicate to routers that these packets are correlated and should be all forwarded or all dropped.

3.1 Goals

Super-packets must ensure that if a super-packet is denied, all subsequent packets from this super-packet are dropped. However, if the super-packet is accepted, the router must make sure that it keeps enough buffer space to ensure forwarding of the subsequent packets in the super-packet. Our algorithm also needs to maintain the same drop rate for regular packets and super-packets to allow decoupling of TCP throughput from the MTU. Its implementation must be efficient: packet processing time must be small and constant to allow packet processing at line speed. This implies some subtleties regarding how super-packet related state is stored on the router to allow super-packets to have a given lifespan without using timers.

Super-packets were designed to function in the wide area or the multi-tenant datacenter, meaning that end-hosts can use any networking stack and are free to choose to use super-packets or not. For this reason, super-packets must be able to properly share the wire with regular packets and any standard TCP implementation. We must also ensure that the super-packet admission algorithm is strategy-proof and does not open up new vectors of attacks. Finally super-packets must be incrementally deployable.

3.2 Super-packets on the wire

Since super-packets appear on the wire as separate IP[1] packets, we must add some information in each of these packets to allow routers to distinguish different super-packets. This is achieved by using the IP *Fragment ID* and *Terms of Service* fields[2]. The fields required for super-packets are the following:

- *Super-Packet Identifier*, a unique identifier chosen by the sender to allow routers to distinguish super-packets. A super-packet identifier must be unique per IP source and destination address pair.

[1] We currently only support IPv4, in this paper IP denotes IPv4. An IPv6 implementation would be designed similarly.

[2] We make the assumption that packets are not fragmented and that these fields are unused. This may not be a safe assumption in an uncontrolled environment.

- *Super-Packet Length*, the length of the super-packet, in packets.
- A *first* flag, notifying routers that this packet is the first one of an incoming super-packet. This flag triggers an admission decision in routers, whereas subsequent packets without it abide by the decision made for the first one.
- A *last* flag, notifying routers that this packet is the last one of a given super-packet. This is meant to be used when a sender realizes that it will send less packets than initially agreed upon, and allows the router to free the corresponding reservation.

IP Terms of Service field							
0	1	2	3	4	5	6	7
first	last	SP Length					

IP Fragment ID field															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
reserved				SP Identifier											

3.3 Super-Packet Admission Algorithm

Super-packets require changes in how routers decide when to drop a packet or not. For this reason, we developed the following algorithm that dictates what decision a router must make when it receives any kind of packet. Simply put, instead of only basing the admission decision on queue occupancy, routers now keep state of which super-packets it accepted and will only use this state information when subsequent packets from a super-packet are received. This state is stored in a hash table and keyed on the source and destination IP addresses and the super-packet identifier. Algorithm 1 is run each time a packet is received, and determines if the packet is dropped or enqueued. Regular packets are treated as super-packets of length 1. To ensure we can uphold our promise to deliver all subsequent packets from an accepted super-packet, we keep a counter of how many packets we must be able to forward, and use this counter to decide on new super-packet acceptance. This can be seen as reserving 70% of the super-packet length in the queue for that purpose. We determined experimentally that a value of 70% is the best trade-off between ensuring promises made to previous super-packets and accepting new ones. Because of this 70% factor, these promises are not absolute guarantees. Super-packet routers can only ensure that they will do their best.

$key \leftarrow srcIP, dstIP, spIdentifier$
if regular packet or first packet of SP **then**
    $queueSpace \leftarrow queueSize - queueLength$
    $weightedSpReserved \leftarrow spReserved \times 0.7$
    if $queueSpace - weightedSpReserved \geq maxSpLength$ **then**
        $packetsLeft \leftarrow spLength - 1$
        $spReserved \leftarrow spReserved + packetsLeft$
        $hashTable.add(key, packetsLeft)$
        $acceptPacket()$
    else
        $dropPacket()$
else
    if hashTable.contains(key) **then**
        $packetsLeft \leftarrow hashTable.get(key)$
        $packetsLeft \leftarrow packetsLeft - 1$
        $spReserved \leftarrow spReserved - 1$
        if $packetsLeft = 0$ **then**
            $hashTable.remove(key)$
        if last **then** ▷ This is only useful when the sender knows they will not be sending the entire SP length, they signal this by setting the last flag.
            $spReserved \leftarrow spReserved - packetsLeft$
            $hashTable.remove(key)$
        $acceptPacket()$
    else
        $dropPacket()$

Algorithm 1: Super-Packet Admission Algorithm

To allow super-packet state to timeout in an efficient manner, the hash table is implemented using two tables, and every epoch the contents of the first are copied onto the second and the first is destroyed, guaranteeing all stored state to be at most two epochs old, and any state deleted this way to be at least one epoch old. This allows us to remove reservations for incomplete super-packets that will never be completed — e.g., when the sender shuts down mid-transmission or if there are drops before the super-packet router. In practice, an epoch is set as twice the largest RTT observed for flows on the router. Since the admission decision in this algorithm does not depend on the length of the super-packet, the drop rate is ensured to be the same between regular packets and super-packets. This algorithm therefore changes the granularity of drops to the super-packet level, efficiently decoupling the throughput of TCP from the size of an individual packet. Super-packets can be incrementally deployed, since routers not aware of super-packets will still route packets normally, and both end hosts do not need to support super-packets to allow gains. Performance improvements will be noticed as soon as the sender and a router responsible for drops are aware of super-packets. Gains will be optimal if all senders and routers responsible for drops are aware of super-packets. If a router responsible for drops does not support super-packets, super-packet properties are not guaranteed and can be detrimental if the first packet of a super-packet is dropped, for example. Super-packets are strategy-proof, since we keep state of how many packets each super-packet is still allowed to send. Users sending less packets than agreed upon will simply waste resources given to them.

4 Implementation

Super-packets were implemented using the Click Modular Router [2]. Click is a software architecture for building customizable routers. It runs inside the Linux kernel and can replace Linux packet forwarding logic. A Click router is a graph of routing elements that each performs a distinct task (packet filtering, route lookup, ARP queries, ICMP error reporting, queuing, etc.). Our super-packet forwarding logic is constructed as a Click element that is positioned before every outgoing interface queue. This element can access the current state of the queue, store state and then is charged with making a decision for each packet it sees: either forward it by sending it to the queue or dropping it. For our implementation of super-packets, we used the latest stable version of Click (version 2.0.1).

Once the Click router has determined an IP packet should be sent out on a given output device (using multiple elements represented here as $Click\ Routing$

$Click\ Routing\ Logic \rightarrow$
$ARPQuerier(10.1.2.2,00{:}11{:}22{:}33{:}44{:}55) \rightarrow$
SPManager$(out2, ...) \rightarrow$
$out2 :: Queue(100) \rightarrow$
$ToDevice(eth4)$

Partial forwarding path of one output device in the Click configuration file.

$Logic$), the packet will go through an $ARPQuerier$ element to receive an appropriate ethernet header, will then be buffered in a $Queue$ element and finally transmitted when it reaches the $ToDevice$ element. Adding super-packet functionality to a Click router only requires adding one custom element, the $SPManager$, and placing one instance of it right before the $Queue$ corresponding to each output device. The $SPManager$ element needs a reference to the queue it is monitoring, since the admission algorithm needs to know the current occupancy of the queue. This element mostly implements Algorithm 1. When it accepts a packet, it simply outputs it into the queue, otherwise the packet is dropped by the $SPManager$ and never reaches the queue. One of the downsides of this implementation is that routers must keep per-super-packet state — in practice a hash table of how many packets are left per super-packet. There is no way around keeping per-super-packet state, since we must be able to know which super-packets were accepted. However, we can avoid keeping track of the number of packets remaining in a given super-packet and only save the number of remaining packets per end host, and then keep a Bloom filter of accepted super-packets. A Bloom filter is an acceptable replacement for a hash table since false-positives in the Bloom filter could only cause more packets to be allowed than agreed upon, which is acceptable if the false-positive rate remains low. Keeping track of remaining packets per host in a separate hash table prevents malicious senders. This reduces the amount of state kept, but at the cost of added complexity.

In order to use super-packets, we needed not only to change how routers buffer packets but also give hosts a way to create super-packets, i.e., to select packets they group together as a super-packet. We have seen that Algorithm 1 requires the router to reserve space when super-packets are accepted, and release this reservation as the subsequent packets of the super-packet are received. Reducing the delay between packets of a super-packet is crucial to the performance of the router, since an outstanding reservation belonging to one flow can harm other simultaneous flows. For this reason, the optimal way of creating super-packets in the context

of a TCP stream is to send each window as a super-packet, since in practice TCP sends bursts of traffic when drops are rare. However, since we must send the super-packet length along with the first packet, we must have an *a priori* notion of how long it will be. This is not achievable in practice, because the length of a outgoing window will depend on the number of TCP acknowledgments received regarding the previous window, and the first packet of the new window will be sent before all acknowledgments have been received, causing us to have no certain way of knowing the length of the window as the first packet is transmitted. To circumvent this, we chose to buffer outgoing TCP packets at the sender and, once we know how long the window will be, it is sent as a super-packet. This is achieved by buffering packets on a per-flow basis and, anytime we experience a given time T_{buffer} during which no packets from this flow are received, we transmit all buffered packets from this flow as a super-packet. We set T_{buffer} to be the time required to transmit five packets; for a $1Gbps$ link with an MTU of 1500 bytes, this is $60\mu s$. Our experiments have shown that this value is large enough to avoid interleaved flows cutting windows in half and small enough to not significantly harm TCP performance. It may seem contradictory to buffer packets in a latency-reducing scheme, however in our case latency is caused by drops which amount to delays in the order of a RTT or even a retransmission timeout — both of which are orders of magnitude greater than our buffering delay.

5 Evaluation

In this section we evaluate super-packets through experiments between real end-hosts using Linux machines running a standard TCP implementation and other machines running the Click Modular Router as routers. We also demonstrate super-packet properties via simulation using Network Simulator 2.

5.1 Simulation separating throughput from MSS

We described in Section 2 that super-packets can remove the direct correlation between MSS and TCP throughput. Our simulations show that performance gains are visible in practice. In this simulation, we stage multiple concurrent TCP flows and single out one of these flows. In one case (Figure 1) the chosen flow will have an increased MSS, which would correspond in practice to Jumbo Frames. In the other case (Figure 2) the chosen flow will be using super-packets.

We observe that not only in both cases the chosen flow receives significant throughput improvements, gains are also comparable between the super-packet flow and the jumbo-frame flow, demonstrating how super-packets have the same effect as increasing the MSS without modifying the MTU.

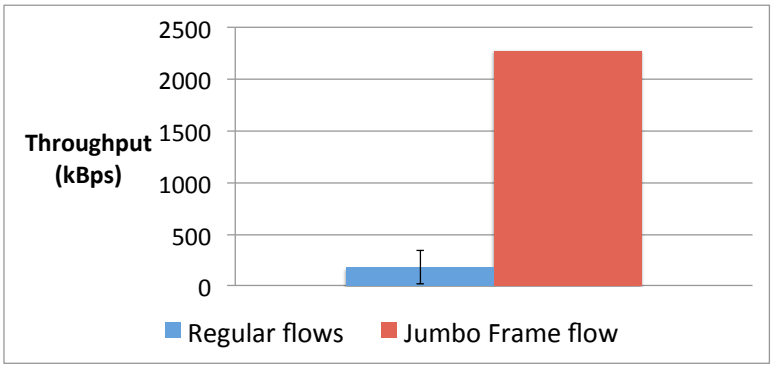

Figure 1: Throughput of 49 regular TCP flows and one TCP flow using Jumbo Frames

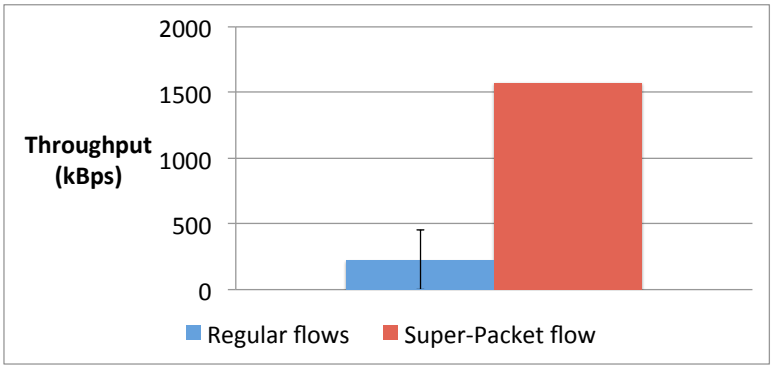

Figure 2: Throughput of 49 regular TCP flows and one TCP flow using Super-Packets

5.2 Hardware and Performance

Our experiments are constructed on three nodes: A, B and C with links A — B and B — C. Node A sends traffic to node C through node B; node B is a super-packet router. Every node has the following characteristics: a Dell PowerEdge

1850 chassis, dual 3GHz Intel Xeon processors, 2 GB of RAM, one 36Gb 15k RPM SCSI drive and multiple Intel Gigabit experimental network ports. Since Click intercepts packets before they reach processing by the Linux kernel and also allow polling of packets from the Network Interface Controller (NIC), we must run a patched kernel and NIC drivers. Machine B runs Ubuntu 8.04 with a patched 2.6.24.7 Linux kernel and patched NIC drivers e1000-7.6.15.5-NAPI. Machines A and C run a vanilla Ubuntu 12.04 LTS with an unmodified 3.2 Linux kernel.

The Click router causes a very slightly heavier load on the machine than the regular Linux kernel forwarding logic. According to [2], Click suffers a 10% performance penalty in packet forwarding. We performed stress-testing on our router on node B. When using packets of size 1500 bytes, we were able to forward 90000 packets per second without difficulty using Click and our super-packet logic — this corresponds roughly to 1Gbps, the throughput of our network controllers. However, when using packets of length 64 bytes, the bottleneck becomes the computation speed of our router, since we do not fully exploit Click parallelism. At this packet length, the Linux kernel can forward 400000 packets per second (approx. 200 Mbps), a regular Click router can only handle 350000 packets per second (approx. 175 Mbps) and a Click router with super-packet logic can reach a rate of 200000 packets per second (approx. 100 Mbps). These rates are lower than the exact maximum rate of packets we can handle since we only tried to observe the order of magnitude we could handle, not the exact rate before failure. In practice our super-packet logic could be implemented in hardware to avoid the overhead of using a modular system such as Click.

5.3 Throughput during incast

Incast is a phenomenon where a great number of simultaneous TCP flows send data through a bottleneck link and start experiencing congestion collapse due to high drop rates. To achieve this scenario, we use three nodes A — B — C and create artificial congestion on link B — C to induce drops at router B. We then send 50 simultaneous flows from A to C and measure their completion times, this was tested using flow lengths of both 10 (Figure 3) and 100 packets (Figure 4).

In Figures 3 and 4, flows are sorted by ascending completion time to allow easier comparisons. The results indicate that super-packets allow a reduction of average flow completion time. Cliff effects are clearly visible in the curves, most likely indicating the presence of retransmission timeouts that are considerably higher than the RTT of flows. The fact that in the experiment with flows of

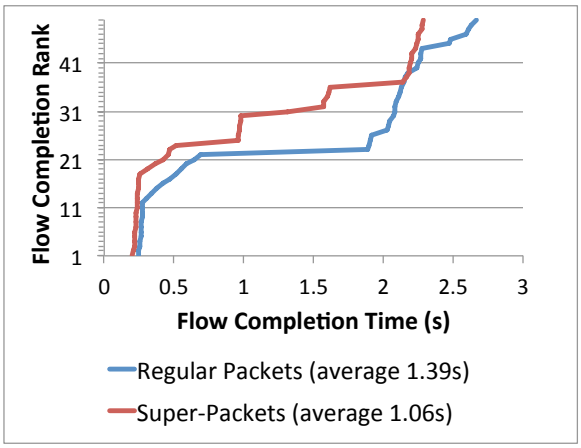

Figure 3: FCT of 50 simultaneous 10-packet-long TCP flows

length 10, the tail still sees improvements thanks to super-packets is interesting, since it seems that these "martyr flows" who had their super-packets denied still seem to do better than without super-packets. This is most likely due to the fact that these flows experienced timeouts, and once they retransmit, the flows in the super-packet case will experience emptier buffers since the other flows completed earlier. This proves that even though we attempted to sacrifice some flows for the greater good, these flows still ended up better off.

We considered avoiding packet buffering and using a fixed super-packet length for all data sent but that performed poorly since the router would reserve space for packets that would never come and had to deny other incoming super-packets that it would have been able to accommodate. Another option was to try to estimate window length but any error in estimation would result in similar inefficiencies and, as explained in Section 4, it is physically impossible to predict the number of incoming packets with absolute certainty.

5.4 Throughput of short flows competing with long flows

We assume, as it is often the case, that the short flows are latency-sensitive, their flow completion times are important, and that long flows are latency-tolerant (throughput being their most important characteristic). This is motivated by the fact that short flows generally have direct consequences, such as element display in a browser or reactions to control messages, whereas long flows, such as video streaming or file transfer, will rely on throughput being sufficient. To induce con-

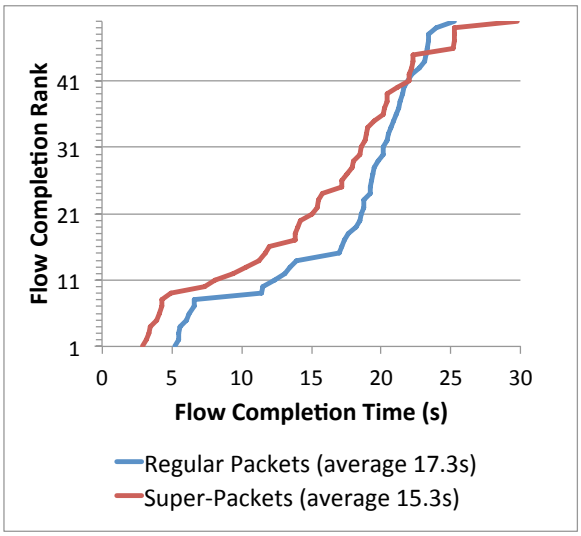

Figure 4: FCT of 50 simultaneous 100-packet-long TCP flows

gestion, we use the same setup as the previous experiment: three nodes, A — B — C, with A sending data to C while link B — C is artificially congested. In this scenario, node A will start 25 infinite flows to C and wait five seconds to allow TCP to increase its congestion window to use all the available bandwidth. Then A sends ten bursts of two short 10-packet-long flows to C, sending each burst as soon as the previous one has completed. We then measure the flow completion times of the short flows and the throughput of the long flows. We run this experiment in three flavors: with regular packets, with super-packets on all flows, and with super-packets only on the short flows.

Without super-packets, some of the short flows experience drops due to the presence of long flows using all available bandwidth and filling the queue in the router. All these flows take longer to complete and some of them even take orders of magnitude longer due to retransmission timeouts. Enabling super-packets mitigates this effect. The gains from super-packets are best when only the short flows use them, since in our current implementation packets and super-packets have the same drop rate, giving a clear advantage to flows using super-packets. The initial congestion windows of the short flows therefore have much better odds of not being dropped, causing the flows to complete rapidly. However, even if all flows use super-packets, the short flows are still protected from the presence of long flows, since their drop rate is still lowered, and they send much fewer packets than the long flows, ensuring that their probability of experiencing drops is

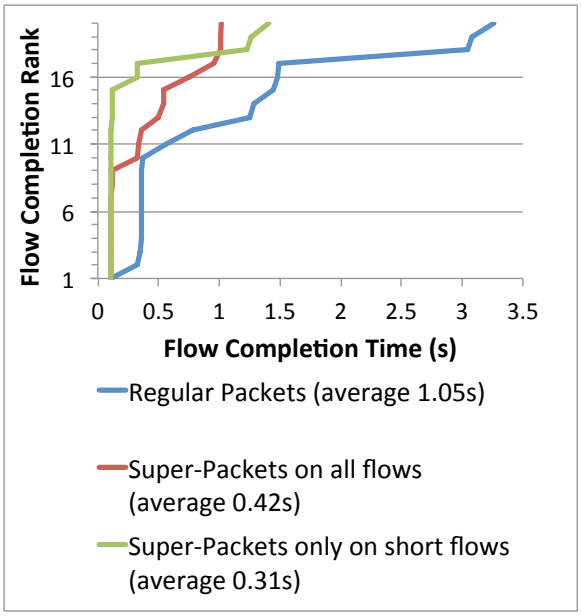

Figure 5: FCT of short TCP flows in the presence of long flows (sorted in ascending FCT)

very low. We still observe that some short flows have their super-packets dropped and experience timeouts, but their drop rate is still much lower than with regular packets, ensuring that their completion times are still improved. It is also interesting to notice that this mechanism does reduce the throughput of the long flows, since reducing drops for short flows implies dropping more packets for the long flows. However the throughput reduction caused by super-packets is lower than 1% (Figure 6) and we believe this to be a worthwhile tradeoff.

6 Related Work

There have been many efforts to try to reduce TCP latency, many of which concentrate on protecting latency-sensitive short flows from long flows only dependent on throughput. Some of these approaches are orthogonal to ours, and many others were developed targeting the single-tenant datacenter environment, often requiring that their new protocol be either alone on the wire, or reliant on trusted input from end hosts.

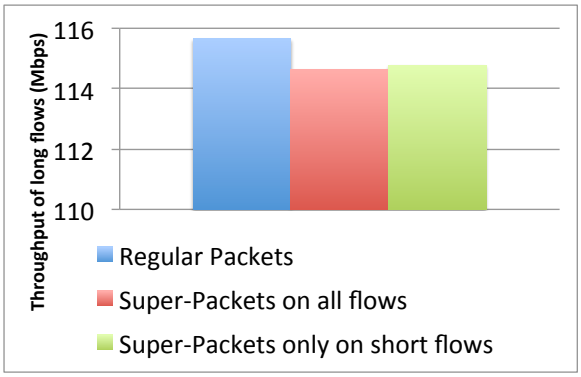

Figure 6: Throughput of long TCP flows in the presence of short flows

6.1 Systems requiring isolation

A considerable amount of literature concentrating on datacenter networking has yielded protocols that only give benefits if they are alone on the wire and used as a complete replacement for TCP. These protocols fail if other endpoints use TCP through the same switches, making them unusable outside of the controlled datacenter environment.

- Data Center TCP (DCTCP) [9] lowers short flow latency by reducing switch queue occupancy. DCTCP leverages Explicit Congestion Notification (ECN) to extract more information about the network: switches mark packets if the current queue occupancy is over a given threshold and the TCP sender reacts to the fraction of marked packets to reduce the window size. This has been shown to improve performance if all endpoints use DCTCP, but if some decide to use TCP, they will gain better throughput than the DCTCP senders and negate all DCTCP benefits by increasing switch queue occupancy. This makes DCTCP unsuitable for any network with uncontrolled endpoints. DCTCP also cannot address a case of severe incast where initial windows from each flow are enough to overwhelm the switch.

- High-bandwidth Ultra Low Latency (HULL) [10] builds on top of DCTCP by also changing the switches. Instead of marking packets when the queue is over a certain threshold, HULL has a notion of a "phantom queue" which is a counter representing how many packets would be in the queue if the bandwidth was only 95% of the actual link. It then marks all packets sent over 95% utilization of the real link. Its reliance upon DCTCP makes it

unusable in the presence of endpoints using regular TCP.

- The Rate Control Protocol (RCP) [11] is aimed at improving FCT instead of throughput. RCP switches keep a rate value $R(t)$ depending on queue occupancy and incoming traffic and packets carry a rate R_p. Routers update $R_p \leftarrow min(R_p, R(t))$ and the receiver echoes R_p back to sender, who will then send traffic at maximum rate R_p. The advantages of RCP are negated if the switches also handle TCP flows since they will fill buffers.

6.2 Systems requiring trust

Other datacenter-oriented systems require the sender to specify the service they need, by either sending flow size in advance, indicating flow deadlines or requesting priorities. These systems require switches to trust the sender to send correct priorities, and fail if all endpoints prefer to optimize their traffic at the expense of others by only sending high priorities or short deadlines for example. For that reason these systems fail in the presence of endpoints not under the operator's control.

- Preemptive Distributed Quick (PDQ) [12] flow scheduling uses Earliest Deadline First and Shortest Job First to prioritize flows, however they rely on the sender sending deadlines and flow lengths to the switches to ensure correct priorities are given. It is therefore not suitable for an environment containing untrusted endpoints.

- D^3 [13] is a deadline-aware control protocol that is customized for the datacenter environment. They rely on deadline information to prioritize traffic, so they solve a datacenter specific problem and can't be deployable in the presence of untrusted endpoints.

- Differentiated Services [14] rely on Per Hop Behavior information given by the sender, who must therefore be trustworthy.

6.3 Synergetic or orthogonal systems

Even though most of the literature on reducing latency targets datacenters, there have been many efforts targeting the wide area and improving user-experience while browsing. Many of these can be used in conjunction with our system to increase latency gains.

- Google proposed to increase the TCP initial congestion window size to 10 packets [7], this greatly benefits from super-packets since we help ensure that all 10 packets are safely delivered. In a traditional browser workload where many short TCP flows are sent simultaneously, super-packets concentrate drops to a small number of flows and ensure minimal latency for most flows.

- TCP Fast Open [8] improves the latency of TCP's three way handshake by sending payload during the handshake, however since most web objects require at least several packets worth of data, it would still benefit from the use of super-packets.

- Considerable work has been put into improving web latency at the application layer. One of the most used examples is SPDY [15], a candidate for HTTP 2.0 that uses header compression, request multiplexing, pipelining and prioritization. Another is the use of Content Distribution Networks (CDN) that cache highly demanded content closer to the edges of the network. Both of these still rely on TCP to send data to a browser and would see gains from super-packets.

- Jumbo Frames increase the physical MTU and can increase TCP performance since they increase the MSS. However they also reduce multiplexing efficiency and can still benefit from super-packets to further increase throughput.

7 Discussion and Further Work

These results indicate that our intuition was correct: allowing routers to base their admission decisions on the context of individual packets can help improve performance of the flows involved. Additionally, super-packets can help mitigate common network deficiencies such as incast or long flows starving latency-sensitive short flows.

However a deeper mathematical analysis would be needed to understand the intricacies of how super-packets affect flows. Figure 3 indicates that in some cases, all flows are positively affected by super-packets. This is counter-intuitive since one might imagine that the rare flows that had their super-packets denied might experience worse performance than when not using super-packets. We be-

lieve that by the time the last flows are retransmitting, the other flows have taken advantage of super-packets and there is less traffic on the link.

The current version of the admission policy algorithm enforces an equal drop rate for super-packets of all lengths and regular packets. This can lead to users only sending super-packets of the maximum length to obtain optimal per-packet drop rates. We have implemented a version of the algorithm that bases its admission decision on the length of the super-packet, but more experimentation is needed to prove that it possesses the same properties as the original algorithm, notably the decoupling of throughput and MSS. We also reason about super-packet lengths in terms of packets, not bytes, because queues and traffic shapers in the Click Modular Router measure occupancy and rates in terms of packets. Moving away from Click would allow us to count in terms of bytes, which is closer to how actual routers reason about their queues.

Our current admission algorithm is based on reserving space for accepted super-packets, using a fixed ratio of how much space we actually reserve compared to how many packets we are expecting. This ratio has been determined experimentally but it may not be optimal in all experienced workloads. We have tested other queueing algorithms, notably one based on RED [16] that only accepts new super-packets under a given threshold. Our results were not conclusive because the router would not be able to fulfill its commitment to forwarding some accepted super-packets. An algorithm based on these combined approaches could lead to improved super-packet performance.

One notable downside with the current implementation is that it is vulnerable to denial-of-service attacks. One malicious super-packet sender could send multiple first packets without ever sending the rest of the super-packet, causing the router to reserve space for long periods of time. If we assume that senders do not spoof their address, we could implement access control and penalize senders that do not use their reservations promptly. However, the current implementation is not safe enough to be used in the wide area.

In conclusion, super-packets have proven to leverage contextual information to improve TCP performance. However the gains we observed are not significant enough to warrant implementing this new scheme, especially since it requires changes to routers inside the network. We are convinced that the concept is promising but not usable in practice until a different take on the algorithm allows significantly better results and all security concerns are addressed.

8 Acknowledgments

We would like to thank Ion Stoica, Ali Ghodsi, Ganesh Anantharyanan and Junchen Jiang for their prior work on the theory of super-packets and their valued opinion on theoretical aspects. We would also like to thank George Li and Thurston Dang for their contributions to the super-packet project. We especially want to thank David Zats for his expertise and advice regarding the use of the Click modular router and his knowledge of the related work. Finally, we'd like to thank Philip Reames, Sylvia Ratnasamy and Daniel Haas for their valued feedback.

References

[1] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: a simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.*, 28(4):303–314, October 1998.

[2] E. Kohler and R. Morris and B. Chen and J. Jannotti and M. F. Kaashoek. The Click Modular Router. In *ACM TOCS*, 2000.

[3] R. Kohavi and R. Longbotham. Online Experiments: Lessons Learned. In *http://exp-platform.com/Documents/IEEEComputer2007OnlineExperiments.pdf*, 2007.

[4] `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html`. Cisco Visual Networking Index: Forecast and Methodology, 2011-2016.

[5] F. Dobrian *et al.* Understanding the Impact of Video Quality on User Engagement. In *ACM SIGCOMM*, 2011.

[6] Matthew Mathis *et al.* The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. In *ACM SIGCOMM*, 1997.

[7] N. Dukkipati *et al.* An Argument for Increasing TCP's Initial Congestion Window. In *ACM SIGCOMM CCR*, 2010.

[8] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. Tcp fast open. In *CoNEXT*, 2011.

[9] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). *ACM SIGCOMM Computer Communication Review*, 40(4):63–74, 2010.

[10] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Proceedings of USENIX NSDI conference*, 2012.

[11] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review*, 36(1):59–62, 2006.

[12] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.

[13] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 50–61. ACM, 2011.

[14] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. An architecture for differentiated services. 1998.

[15] Bryce Thomas, Raja Jurdak, and Ian Atkinson. Spdying up the web. *Communications of the ACM*, 55(12):64–73, 2012.

[16] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397–413, 1993.