Enabling Router-Assisted Congestion Control on the Internet

Marcel Flores, Alexander Wenzel, and Aleksandar Kuzmanovic
Electrical Engineering and Computer Science Department
Northwestern University
{marcel-flores alexanderwenzel2017}@u.northwestern.edu akuzma@northwestern.edu

***Abstract*—Enabling communication between routers and endpoints has long been sought after as an approach to congestion control in the Internet. However, the narrow-waist of TCP/IP has complicated the deployment of such communication. In this paper, we present Kick-Ass[1], a congestion control mechanism that enables explicit rate congestion control protocols to be deployed within the TCP/IP stack. The key idea is to utilize packet lengths as a vehicle to communicate fine-grained explicit rate and other information from routers to endpoints and vice versa. Given that our approach (*i*) requires no explicit coordination among Kick-Ass routers, (*ii*) no explicit coordination among Kick-Ass routers and endpoints, and (*iii*) is effective on paths that include legacy routers, it provides a practical road towards a faster Internet, today.**

Using large-scale simulations, testbed experiments, and wide-area Internet evaluations, we demonstrate that (*i*) a basic explicit-rate protocol using the Kick-Ass mechanism improves flow completion times by up to an order of magnitude and outperforms endpoint-based approaches, including CUBIC and PCC. (*ii*) Kick-Ass is incrementally deployable on the Internet. (*iii*) Deploying Kick-Ass at end-hosts and edge routers can enable the above performance benefits, without waiting for universal adoption. (*iv*) Our packet-fragmentation mechanism is well behaved on the Internet.

I. Introduction

A fundamental question in congestion control is when should an endpoint transmit each packet of data. An ideal scheme would transmit a packet whenever capacity to carry a packet is available [1]. Still, estimating and utilizing the available capacity from an endpoint is an extremely difficult task for numerous reasons: the existence of many concurrent distributed senders, variable network and queuing delays, dynamic flow arrivals and departures, and multiple congested points, to name a few.

Necessarily, getting help from the network in resolving this difficult task, *i.e.*, from routers that directly observe the network traffic and its dynamics, has historically been recognized as a viable idea. In today's Internet, such mechanisms include Active Queue Management (AQM) schemes such as RED that help smooth the traffic and reduce traffic oscillations [2]. Another example is Explicit Congestion Notification (ECN) that uses a single bit in the IP header to signal network congestion to the endpoints without dropping packets [3]. While certainly useful, such mechanisms stand at the *low-end* of possible performance benefits that would be achievable if routers were able to send more fine-grained than single-bit notifications to the endpoints [4].

It has been demonstrated that even a single additional bit would bring significant performance improvements [5]. Having additional *bytes* of data per packet would enable explicit rate protocols such as XCP [4] or RCP [6], which can achieve additional dramatic benefits. These include the decoupling of utilization and fairness [4], substantial throughput increases particularly in large bandwidth-delay product and data center networks [6], [4], virtual removal of queuing-delay latencies [6], [4], *etc.* Unfortunately, to the best of our knowledge, such protocols have never been deployed on the Internet due to their inherent incompatibility with the TCP/IP stack. Traditional routers simply have no mechanism for communicating complex information directly with endpoints.

In this paper, we present a congestion control mechanism that enables advanced router-assisted congestion control schemes, including explicit-rate methods, to be deployed *within* the TCP/IP protocol stack. Our key contributions are the following: (*i*) we enable a simple and practical pathway towards a realistic wide-scale deployment of explicit-rate congestion control protocols on the Internet, (*ii*) we demonstrate, via simulations, test-bed and Internet experiments, that a basic explicit-rate protocol can achieve substantial performance gains, *i.e.*, flow completion times decrease by up to an order of magnitude, network utilization approaches the full capacity, while queuing delays become negligible. (*iii*) This is possible to achieve in today's Internet even in partial deployment scenarios, *i.e.*, flow completion times decrease by up to 4 times in our Internet experiments, yet without any side-effects for those who do not support the approach. (*iv*) Beyond congestion control, our contribution lies in demonstrating that embedding information into existing protocols can be a powerful approach for overriding rigidity of the Internet (beyond IP).

The key idea behind Kick-Ass is to utilize IP *packet fragmentation* as a vehicle to communicate fine-grained explicit flow rate information from routers to endpoints. In particular, the explicit rate information is implicitly embedded in the size of the first fragment of a packet fragmented by a router. The smaller the fragment size, the smaller the explicit rate communicated to the endpoints. Due to the monotonic nature of the minimum rate along a path, it is guaranteed that the right explicit rate information is communicated to the endpoints

[1]Kick-Ass is an open-source project; Linux and NS-3 code are available at the author website: http://networks.cs.northwestern.edu/projects/kickass/

without *any* explicit coordination among routers. The lack of explicit coordination required among Kick-Ass-enabled devices further dramatically eases incremental deployment.

We design Kick-Ass to cope with the significant heterogeneity of networks on the Internet, which range from low-rate wireless networks to high-rate data-center networks. We derive an appropriate encoding scheme which encompasses 5 orders of rate magnitude. In addition, contrary to native explicit-rate protocols, our key goal in the endpoint design is to enable Kick-Ass endpoints to effectively interact with the legacy TCP traffic without affecting their performance.

We initially evaluate Kick-Ass in the NS-3 simulator [7]. We stress our protocol in a number of challenging scenarios and demonstrate its ability to substantially improve the performance in all evaluated scenarios. In particular, in full-deployment scenarios, Kick-Ass is able to achieve the performance of native explicit-rate protocols, while performing only an RTT slower than native methods which use explicit headers to communicate rate information. In partial deployment scenarios, we show that Kick-Ass can quickly adapt to network dynamics, *i.e.*, it can detect Kick-Ass-enabled bottlenecks and vice versa and adjust the rates over short time scales.

To demonstrate Kick-Ass' practicality and evaluate its efficiency, we implement a Linux-based Kick-Ass router and endpoints using existing network hardware. Our test-bed experiments show Kick-Ass outperforms PCC [8] and CUBIC flow completion times by $2\times$ in low load environments. We further demonstrate that it can provide 0 ms median queuing delay in high load environments, and outperforms CUBIC and PCC. Next, we deploy Kick-Ass endpoints on the wide-area Internet. Our Internet evaluations show that the simplest possible partial Kick-Ass deployment, with endpoints and a single Kick-Ass edge router, brings significant benefits.

Finally, we conduct a measurement study to understand the state of the IP fragmentation in today's Internet. The study, which involved 405 clients from 62 countries, found no signs of discrimination against fragmented packets by network routers. We found that various middle-boxes are responsible for dropping about 8% of connections that involve fragmented packets. To cope with such cases, we ensure that Kick-Ass routers respect the Do-Not-Fragment field in IP headers, thereby enabling communication towards such destinations. We also show that Kick-Ass is deployable within IPv6. Finally, we show that the network and computational overhead associated with fragmentation and reassembly can be dramatically reduced, further demonstrating the viability of Kick-Ass as a generally deployable system.

II. Design

Kick-Ass enables the explicit communication between routers and end-hosts by using the implicit signal of packet lengths. We first focus on the way the rate information is communicated from routers to endpoints. Kick-Ass consists of two main components: *(i)* the router, which determines the allowed rate for senders on each of its links and communicates them, and *(ii)* the endpoints, which receive these communications and adjust their sending rates accordingly. It is important to keep in mind that Kick-Ass is a mechanism, enabling the implementation of router-based congestion control algorithms by allowing the router to communicate sending rates to endpoints. The particular algorithm used to determine the rate is generic.

Figure 1 depicts this communication process. The process begins with a sender sending packets as it would in a normal TCP connection. During the SYN/ACK phase, the sender and receiver agree to use Kick-Ass by way of TCP options. The sender then begins sending data into the network. When packets arrive at the first Kick-Ass router, Router A in the figure, the router performs its explicit-rate calculation algorithm, determining the appropriate rate. As the packets depart, they are fragmented, setting the length of the leading fragment to a value which encodes the rate. A larger leading fragment indicates a higher rate, while a smaller leading fragment indicates a lower rate. We discuss the specific mechanism for encoding rates to packet lengths in more detail in Section II-A

At downstream Kick-Ass routers, Router B in the figure, the router calculates its rate and the corresponding leading fragment size. If the size of a received packet is less than the computed size, indicating an upstream router has selected a lower rate, the router simply routes the packet normally. If the size of the received packet is larger, or the packet does not yet indicate a rate, it is fragmented to the correct length. Only the leading fragment is ever set to size: fragments of offset greater than 0 are routed normally, regardless of size. Embedding information in non-leading fragments is feasible, but beyond the scope of this work. By the time a packet has traversed the entire path, the leading fragment will indicate the rate allowed by the path bottleneck.

When a leading fragment arrives at the receiver, the receiver uses the length to compute the sending rate and packages the computed rate in the body (data payload) of the next ACK it sends. Upon receipt of such an ACK, the sender adjusts its sending rate accordingly, for example by changing the size of its congestion window or via explicit packet pacing. This approach thereby allows a router communication scheme similar to those seen in RCP [6] and XCP [4] without the addition of a new header. Since the information is entirely encoded in traditionally compliant TCP/IP packets, downstream legacy routers which do not support Kick-Ass are able to receive and route packets as normal, without a need for any additional handling or awareness of Kick-Ass. The lack of additional TCP options on data packets or shim layers also decreases the likelihood that various middle-box services will interfere with the communication [9]. In the event that the sender never receives any information from the router, for example there are no Kick-Ass enabled routers on the path, it simply continues to behave as it would in traditional TCP.

The use of the length of only the first fragment provides us with several advantages. First, the router need only receive the first fragment in order to indicate a rate. Since fragmented packets are routed independently, they are not guaranteed to pass through the same routers. Waiting for trailing fragments could introduce significant delay. Since a router only needs to

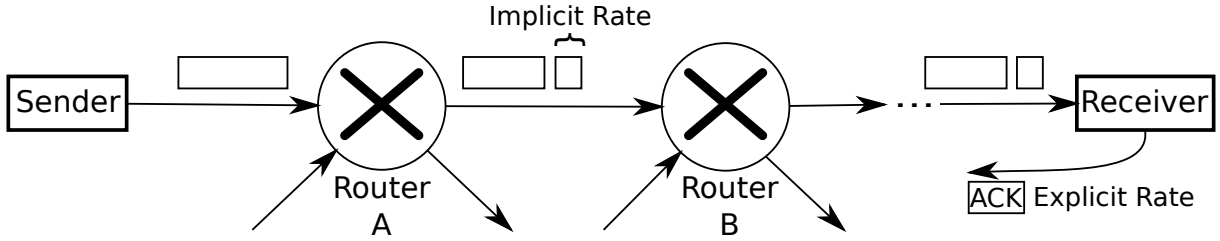

Fig. 1: Kick-Ass in action: Router A communicates a flow rate via packet fragmentation

change the indication of a packet in the case of a decrease in rate, and therefore a decrease in leading fragment length, it is able to further fragment without having any of the later pieces. Third, the receiver is able to determine the rate with the arrival of the first fragment, as the first fragment always contains the TCP header. This allows it to update the rate for the appropriate flow immediately.

Since Kick-Ass routers ultimately behave in a manner which is in accordance with the IP specifications, the fragmenting transformations are safe to perform on traffic which is not operating Kick-Ass on the endpoints. This compatibility with legacy routers *and* flows means that Kick-Ass is ready-made for partial deployments.

A. Kick-Ass Rate Encoding

We now explain how the router uses the leading fragment length to indicate the rate. The design of the IPv4 header specifies the starting point of the packet payload in the reconstructed packet in 8 byte words, restricting the set of fragment lengths to multiples of 8 bytes. Furthermore, data bearing packets are generally $1,500$ bytes long, including the IP header [10]. Kick-Ass is therefore able to fragment to 187 possible lengths. Packets smaller than $1,500$ bytes are ignored and routed normally. If a flow is of sufficient length, a router is likely to see a number of full sized packets per RTT, minimizing the effect of too-small packets. If no suitably sized packets arrive, the system will behave as traditional TCP.

To make the most of this set of indicators, we use a function which maps from sending rates to packet lengths, *i.e.* a function F that tells us what length L our leading fragment should be to indicate a rate R. The nature of congestion control is that we would like this function to provide greater precision at lower rates, at the expense of precision at very high rates. We therefore select a logarithmic function.

On the lower end, we would like the system to have a minimum sending rate of 1 packet per round trip time (RTT). We therefore choose as a lower rate 120 Kbps, providing a single packet per RTT for a sender with a 100 ms RTT sending 1500 byte packets. If a router requires even lower rates, endpoints switch to legacy TCP mode, which provides such capabilities. In particular, when a flow's rate is less than the minimal encodable value, the KA router stops fragmenting packets, which pushes the endpoint into TCP mode, as described in Section II-C2. On the upper end, we allow the function to encode rates of up to 10 Gbps. Since we are operating on RTT time scales, we take our rate R to be in bytes per millisecond. Solving for a logarithmic function that would map a length of 24 bytes to our lower limit and 1480 bytes to our upper limit (excluding the IP header), we find our function F to be:

$$L = F(R) = 128 * \ln\left(\frac{R}{12.4392}\right) \text{ bytes.}$$

The computation can be reversed by simply taking the inverse function of the packet length.

A minimum payload of 24 bytes allows us to ensure the leading fragment always contains a TCP header, a common requirement for many middle-boxes. The first 1000 bytes of the packet encode rates only up to 245 Mbps, while the remainder covers up to 10 Gbps, providing greater granularity at lower rates. While we have selected this particular range, it is reasonable to expect that any final determination of a standard for Kick-Ass could trivially adjust this function, allowing a greater, or more specific, range of possible sizes.

B. Kick-Ass RTT Encoding

In addition to communicating rate from routers to endpoints, router-assisted congestion control algorithms require communicating RTT information from endpoints to routers, *e.g.*, [6]. To do so, Kick-Ass again uses packet length. Every 100 ms, the endpoint sends a packet of an *indicator size* with the Do-Not-Fragment bit set. When the router receives such a packet, it looks up the corresponding RTT and sends the packet without fragmenting it. In order to minimize regular packets being taken for indicator packets, our *indicator sizes* are 600 to 611 bytes, where 600 indicates a 1 ms RTT, 601 the next 2ms (2-3 ms), 602 the next 4ms (4-7ms), exponentially up to 611 indicating an average RTT greater than 2048 ms. The router then takes the RTT to be the midpoint of the indicated range. This allows endpoints to communicate a range of RTTs, focusing accuracy on lower RTTs, where precision is more important. We evaluate the safety of this approach in Section IV-C below.

C. Kick-Ass Endpoint Design

1) Kick-Ass flows at a Kick-Ass Router: We first consider a full-deployment scenario. In terms of Figure 1, we consider that both routers A and B are Kick-Ass enabled. We further consider that all flows respond correctly to signals from Kick-Ass enabled routers. This represents a model for early deployments: Kick-Ass could be deployed in a datacenter, allowing flows to traverse only Kick-Ass routers. Moreover, an edge network may enable Kick-Ass on endpoints and install Kick-Ass routers. While the flows must traverse the

legacy Internet, the routers themselves features only Kick-Ass traffic. We explore the performance of such a scenario in Section III-C.

2) Mixed flows at a Kick-Ass Router: Here, we consider a Kick-Ass router which is serving both Kick-Ass enabled endpoints and traditional TCP endpoints. In terms of Figure 1, this means there is TCP cross traffic at Kick-Ass enabled Router A. Regardless of the rate computation scheme used at the router, TCP flows will ignore signals from the router, sending at increasing rates, causing queueing, and eventually drops. For many methodologies, this would result in TCP starving out the more intelligent mechanisms which respond to pre-drop congestion signals. However, the change in queue state means a router is able to determine when a portion of the flows it is processing ignore its signals and indicate to its endpoints that they must be more aggressive. This is indicated by the queues exceeding a threshold of 50%. In the event that the queues pass this threshold, the router ceases indicating a sending rate and stops fragmenting packets. At the endpoints, failure to receive a rate signal from the router for more than a RTT indicates that they should revert to *TCP mode*. In TCP mode, senders behave as traditional TCP senders.

By reverting to TCP mode, they are then able to compete for their fair-share of the capacity using traditional methods. When the congestion period ends at the router (indicated by a dip below the described thresholds), the router again begins indicating rates. On receipt of such rates, the endpoints immediately return to Kick-Ass mode. We see that the performance of Kick-Ass is bounded below by TCP: as soon as the router encounters a situation in which it knows it cannot properly handle, it reverts to the compatible traditional methods. This mechanism also allows Kick-Ass endpoints to handle the case in which a route may change and no longer include a Kick-Ass router: the nodes simply revert to TCP. We evaluate Kick-Ass in such scenarios in Section III-D1.

3) Congestion at a Non-Kick-Ass Router: In this case, there is congestion at a router which is not indicating a rate. From Figure 1, this corresponds to the situation in which Router B is a legacy Non-Kick-Ass router and is suffering from congestion. In particular, there is congestion in the network, but it is not reflected in the rates indicated by the routers, *i.e.* router A in Figure 1. In this case, the sender will eventually encounter a traditional congestion signal, *e.g.* a dropped packet, triple duplicate ACK, or ECN bit. Upon the receipt of such a signal, the sender reverts to TCP mode, performing standard TCP back off procedures. While in TCP mode, the sender continues to record information it receives from Kick-Ass routers. Once the senders rate recovers to the rate indicated by the Kick-Ass routers, it returns to Kick-Ass mode, following the indicated rate.

Again, by reverting to TCP in the case of congestion that does not manifest itself in the rate, Kick-Ass is able to bound its performance by that of TCP. This allows Kick-Ass flows to compete fairly with existing TCP flows without starving them of resources at downstream legacy routers. We demonstrate the effectiveness of this technique in Section III-D2.

Summary: If a sender suffers a loss event or stops receiving Kick-Ass signals, it switches to TCP. When the TCP rate grows above the last-received Kick-Ass rate, the endpoint switches back to Kick-Ass mode.

III. Design Evaluation

A. Router Rate Calculation

The chief contribution of Kick-Ass is the enabling of communication between the router and endpoint. It is therefore generic: the router itself can use whatever scheme it wishes in order to determine the rate it would like to indicate to the sender. Likewise the endpoints are flexible, and can use whatever protocol they wish for handling TCP failover mode.

For the purposes of our implementation and evaluations, we implement a version of RCP, the Rate Control Protocol [6]. RCP is an explicit-rate congestion control protocol, in which routers attempt to divide bandwidth evenly among flows by emulating processor-sharing, selecting a single rate for all flows. Specifically, the router measures the input traffic on a time interval t. If the input traffic is below the capacity, all flows are increased evenly. If the input traffic is above the capacity, all flows are decreased evenly. The router also measures the queue size after t time, and sets aside a fraction of the capacity in order to empty the queues.

For TCP failover mode, we use TCP NewReno. We again emphasize that our choice of an algorithm based on RCP and TCP NewReno was not out of necessity: the Kick-Ass mechanism is flexible, allowing a choice of algorithm at the router. The endpoint behavior is likewise flexible.

B. Simulation Setup

We implemented versions of Kick-Ass and RCP in NS-3 [7] to provide a reference. We further compare the performance against TCP NewReno, as it provides a fair comparison with our TCP failover. We generate a workload modeled with a fixed number of on-off senders. Senders begin in off mode, waiting for a time drawn from an exponential distribution. When the time expires, the sender begins a flow of a size drawn from a Pareto distribution and runs until completion. After the flow is finished, it switches to the off state and repeats. All data packets from a sender are 1500 bytes long. Parameters for both the off-time and flow lengths depend on the experiment and are described in each case.

Unless indicated otherwise, the simulations are done on a dumbbell topology in which multiple senders pass through a single bottleneck and communicate with multiple receivers. When testing full-deployment scenarios, the router is of the appropriate type: TCP flows only pass through TCP routers, RCP flows through RCP routers, and so on. The queues are modeled as drop-tail queues.

C. Full-Deployment Scenarios

1) Kick-Ass Dynamics: Here, we evaluate scenarios in which all endpoints and all routers on the path support Kick-Ass. Initially, we explore RCP and Kick-Ass protocol dynamics in a simple scenario with long-lived flows. We set

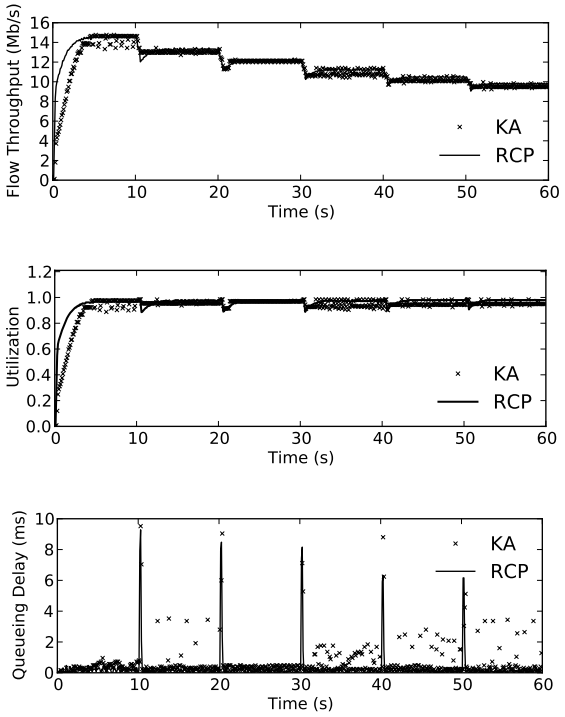

Fig. 2: An additional flow is added every 10s. Both KA and RCP easily adapt and maintain utilization and a fair share of bandwidth.

the link rate to 150 Mbps, queue size to 1000 packets, and the RTT to 100 ms. The simulation starts with 10 concurrent flows. Then, every 10 seconds we add one additional flow.

Figure 2 shows RCP and Kick-Ass performance over time in this scenario. In particular, Figure 2a depicts throughput of a *single* flow, Figure 2b plots the 150 Mbps bottleneck link utilization, and Figure 2c illustrates the queue dynamics. We first focus on the RCP performance.

Figure 2a shows that a single RCP flow quickly converges towards its fair share rate, which is 15 Mbps initially, *i.e.*, 150 Mbps/10. As soon as the new flow arrives at 10 seconds, the flow rate quickly stabilizes at the new rate of 13.63 Mbps (150 Mbps/11). When new flows arrive at times 20 s, 30 s, *etc*, RCP manages to adjust to the fair rate within 1 RTT. Figure 2b demonstrates that other flows in the system show the same behavior, such that the system utilization approaches the full link capacity. Figure 2c shows that RCP is capable of achieving this *without* driving the queue the way TCP is known to do. In particular, RCP creates moderate queue spikes at moments when new flows join. Note, however, that such queuing is promptly reduced, within a single RTT.

Figure 2c shows the Kick-Ass queuing behavior. First, expectedly, the plot shows that Kick-Ass inherits the spikes at multiples of 10 s existent in the native RCP protocol. Second, unlike RCP, Kick-Ass occasionally generates very small queuing delay spikes. These are seen between 10 and 20 s, as well as between 30 and 60 s. We found that these effects happen due to the constant number of flows and smaller resolution at which Kick-Ass can communicate explicit rates to the endpoints, *i.e.*, 187 levels in Kick-Ass relative to the

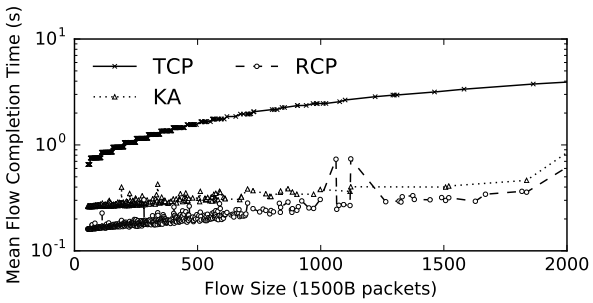

Fig. 3: The mean flow completion time of flows under light load in the isolated case.

integer rate precision in RCP. We find that this effect does *not* get worse with more flows, and it diminishes in more dynamic flow arrival and departure scenarios. Indeed, with large-scale populations the available per-flow bandwidth decreases, hence the resolution of Kick-Ass rates increases.

2) Lightly-Congested Scenario: Next, we evaluate RCP and Kick-Ass in a lightly congested scenario. We set the link rate to 150 Mbps, queue size to 1000 packets, and the RTT to 100 ms. Contrary to the above experiment, we have 32 clients who concurrently send data into the network. Flow sizes are distributed between 1 and 2000 packets. The mean flow size is set to 100 packets, with pareto shape of 1.8 and an exponentially distributed random gap between each flow with a mean of 0.5 seconds. This leaves ample available bandwidth for the endpoints. Prior research has shown that TCP flows indeed spent most of their lifetime in the start-up phase [11]. Others argued that transport protocols need to be optimized for such common scenarios [6], [12].

Figure 3 shows the mean flow completion time for flows between 1 and 2000 packets long. Both RCP and Kick-Ass are able to significantly outperform TCP, completing flows nearly an order of magnitude sooner. Despite an exponentially increasing start-up behavior, TCP lags dramatically behind RCP and Kick-Ass, which manage to effectively utilize explicit-rate feedback from the router. We do note that Kick-Ass completes flows an RTT slower than RCP. This is a result of communicating the rate in the first data packet rather than in the SYN packet, as RCP does. Despite this, Kick-Ass still significantly outperforms TCP. This suggests that in low-congestion scenarios, Kick-Ass provides users with a significant improvement to flow completion times, avoiding the startup costs of TCP.

Other Topologies Finally, we evaluate Kick-Ass in lightly and heavily congested scenarios for a variety of topologies We explore (i) a non-uniform RTT scenario where RTT is varied in the range from 10 ms to 250 ms, (ii) a scenario with five bottlenecks, and (iii) a data-center-like scenario with a 10 Gbps link. In all evaluated scenarios, we confirm that Kick-Ass effectively emulates RCP's performance. We omit details of these experiments here in an attempt to explore partial-deployment scenarios in more depth.

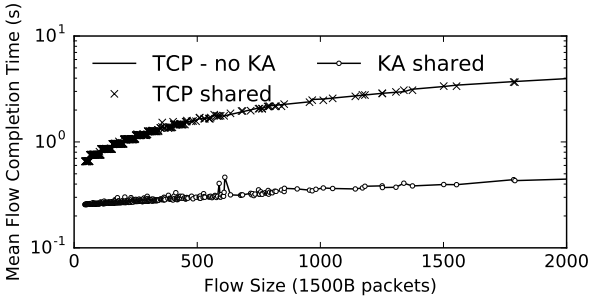

Fig. 4: The mean flow completion time for TCP and KA when running concurrently under light load at a KA router.

D. Partial-Deployment Scenarios

Given that it is impossible to deploy Kick-Ass at all endpoints and routers simultaneously, the question is how Kick-Ass and TCP flows affect each other when they are multiplexed at Kick-Ass or legacy routers. Here, we explore such partial-deployment scenarios in which half of the flows are Kick-Ass-enabled, while the other half are not. First, we explore congestion at a Kick-Ass bottleneck link, then at a non-Kick-Ass bottleneck. In both cases, we evaluate both light and heavy congestion scenarios. Unless otherwise indicated, we set the link rate to 150 Mbps, queue size to 1000 packets, and the round-trip time to 100 ms.

1) Congestion at a Kick-Ass Bottleneck:

Light-Congestion Scenario Figure 4 shows the result of a lightly congested scenario. When TCP flows are alone in the system, the Kick-Ass router computes the fair-share rates and sends this information to the endpoints. However, because the endpoints are not Kick-Ass enabled, they do not understand explicit-rate information embedded in IP fragments. Hence, they apply the generic TCP algorithm and achieve the performance shown in the figure (see curve `TCP no KA`).

Figure 4 shows the performance when Kick-Ass and TCP flows are multiplexed together. The first insight is that TCP's performance is virtually unchanged, despite the presence of Kick-Ass flows in the system. Indeed, there is almost no difference between the `TCP no KA` and `TCP shared` results. This demonstrates that Kick-Ass flows have *no* side-effects for the competing TCP flows in this scenario. Despite the fact that Kick-Ass flows promptly set their rate to the one advertised by the router, such a high rate neither generates excessive queuing nor packet losses. Hence, TCP flows retain their performance. At the same time, Kick-Ass flows (see curve `KA shared`) substantially improve their performance relative to TCP flows, approximately by an order of magnitude.

Heavy-Congestion Dynamic Scenario Here, we evaluate a scenario with long-lived Kick-Ass flows dynamically multiplexed with TCP cross traffic. In particular, there are 64 Kick-Ass flows in the system. In addition, at 20, 60, 100 sec, *etc* we generate an additional 64 TCP flows that send data for 20 sec periods. As a result, we create Kick-Ass only epochs (*e.g.*, 0-20 sec, 40-60 sec, *etc*) and multiplexed Kick-Ass and TCP epochs (*e.g.*, 20-40 sec, 60-80 sec, *etc*). The key goal of this experiment is to evaluate Kick-Ass router and endpoint mechanisms' ability to effectively switch between Kick-Ass and TCP modes of operation.

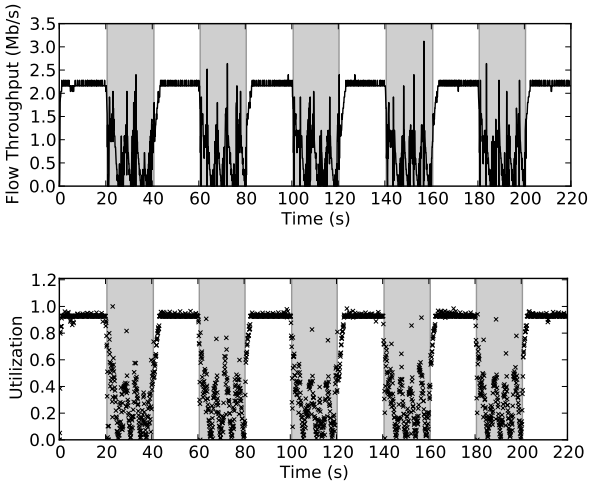

Fig. 5: Kick-Ass and TCP running on a KA router. TCP flows are introduced at 20, 60, 100, 140, and 180 seconds and last for 20s. Shaded regions indicate when the router is in TCP mode.

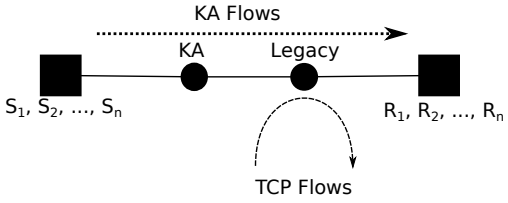

Fig. 6: Congestion at a non-KA router.

Figure 5 shows the throughput of a single flow (Figure 5a) and the link utilization (Figure 5b) over time. Initially, when 64 Kick-Ass flows are present in the system, the Kick-Ass flow rate stabilizes at the fair rate, *i.e.*, approximately 2.34 Mbps, which corresponds to 150 Mbps/64. Likewise, the utilization approaches the full link capacity, and the queuing delay (figure omitted) becomes negligible. Next, at time 20 sec, an additional 64 TCP flows enter the system. Because they are not Kick-Ass enabled, they do not adjust the rate according to signals sent from the router. Necessarily, they thus push the bottleneck queue, causing queuing and packet losses. By design (see Section II-C2), the Kick-Ass router detects excessive queuing caused by non-Kick-Ass flows, and stops sending fair-share rates to endpoints, *i.e.*, it abandons fragmentation. As a result, in absence of Kick-Ass signals from the network, Kick-Ass endpoints enter TCP mode. Thus, the Kick-Ass router avoids starving Kick-Ass flows, and helps them retain their TCP fair share during such epochs. As soon as the heavy-congested TCP-induced epoch passes, the Kick-Ass router reestablishes normal operation. This is demonstrated at time 40 sec. The router quickly detects the absence of the TCP cross traffic and starts sending fair-share rates via fragmentation. Figure 5 shows such periodic behavior where shaded regions indicate when the Kick-Ass router enters TCP mode.

2) Congestion at a Non-Kick-Ass Bottleneck: Next, we consider the case where congestion happens at a non-Kick-Ass router, yet Kick-Ass signals are sent from a non-bottlenecked

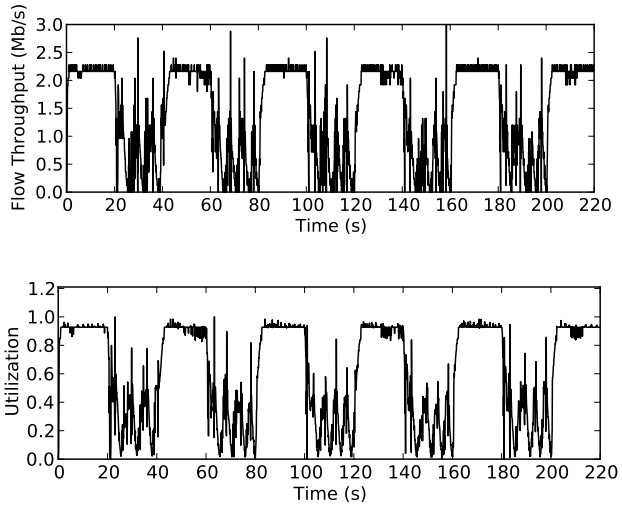

Fig. 7: Congestion at a Non-KA router. TCP flows are introduced at 20, 60, 100, 140, and 180 seconds and last for 20s.

Kick-Ass-enabled router. This is illustrated in Figure 6. The first router is Kick-Ass-enabled, yet congestion happens at the second router, due to the TCP cross traffic. The purpose of this experiment is to evaluate the ability of Kick-Ass endpoints to disregard Kick-Ass signals when congestion happens at non-Kick-Ass bottlenecks.

Light-Congestion Scenario We repeat the above light-congestion scenario. The only difference is that Kick-Ass flows traverse both routers, while TCP flows traverse only the second legacy router. We again find that the flows complete undisturbed, allowing short flow completion times, replicating the performance of the previous case.

Heavy-Congestion Dynamic Scenario Here, we repeat the above dynamic congestion scenario. The only difference is that 64 Kick-Ass flows traverse both routers shown in Figure 6, while TCP flows traverse only the second non-Kick-Ass router in the periods 20-40 sec, 60-80 sec, *etc*.

Figure 7 shows the results, *i.e.*, Figure 7a depicts the rate of a single Kick-Ass flow, while Figure 7b depicts the utilization at the second link. In absence of TCP cross traffic at the second router, Kick-Ass flows effectively utilize the available bandwidth and achieve fairness, thanks to the explicit-rate signals from the first Kick-Ass-enabled router. At time 20 sec, when TCP flows start pushing the queue at the second router, Kick-Ass flows detect this change and switch to the TCP mode, using mechanisms explained in Section II-C3 above. Since the first Kick-Ass-enabled router is not the bottleneck, it continues sending explicit-rate signals throughout the entire experiment. However, congestion at the second router, *i.e.*, at 20 sec, forces Kick-Ass endpoints to abandon such signals and switch to the TCP mode. We verify that Kick-Ass flows and native TCP flows fairly share the available bandwidth in such a case. This is not a surprise given that Kick-Ass endpoints behave exactly as TCP flows in this mode. As soon as the congestion at the second router eases, *i.e.*, at 40 sec, Kick-Ass endpoints promptly move back to the Kick-Ass mode, establishing close-to ideal performance.

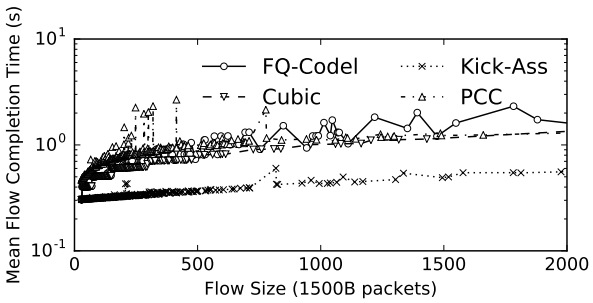

Fig. 8: The mean flow completion time for Kick-Ass, CUBIC, CUBIC on FQ-CoDel with ECN, and PCC during light congestion.

IV. Performance

We examine the performance of Kick-Ass in the real-world. First, we consider an evaluation in a test-bed which allows us full control of the network. We then explore the performance of Kick-Ass on the wide-area Internet. Finally we measure the behavior of fragmented packets on the Internet.

A. Testbed Experiments

Implementation The testbed consists of a set of senders, a router with two interfaces, and a receiver. All are connected via a 3com gigabit switch. The senders and receiver are configured to send all traffic through the router. We use the Linux Netem tool to add 100 ms of delay to the round trip time between the senders and the receiver. We implement the Kick-Ass router on a 3.3Ghz Intel i5 machine running Arch Linux with kernel 3.12.3. Our router is implemented as a Linux queuing discipline. For arriving packets, the queue behaves as a drop-tail queue. As packets are de-queued, they are fragmented to the appropriate length, which is stored in the queuing discipline. We use a modified version of the Linux IP stack to perform the fragmentation to ensure correctness. The endpoints consist of 4 Intel 3.3Ghz Intel i5 machines running Arch Linux.We use a modified version of MultiTCP [13] to pace outgoing packets. Both the pacing delay and the congestion window are updated with the arrival of new rate information from the receiver in ACKs.

Evaluated Schemes For comparison, we consider (i) CUBIC, a widely-adopted TCP variant, which is the default in modern versions of Linux [14], (ii) PCC, a recently-proposed scheme from academia [8], and (iii) CUBIC on FQ-CoDel, which uses packet delay to determine when to indicate congestion, instead of traditional queue length, combined with fair queuing methods [15], [16]. FQ-CoDel further makes use of ECN signals, instead of dropping packets.

Light Load First, we explore the performance of Kick-Ass in the light-congestion scenario. We restrict the outgoing link on the router to 100 Mbps, and model traffic as in simulation, only with 8 total senders.

Figure 8 shows the performance of the evaluated schemes under light load. We see that compared to CUBIC, Kick-Ass

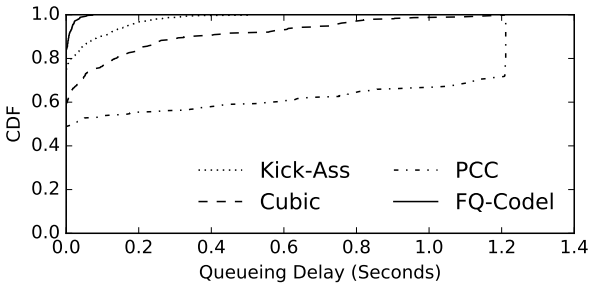

Fig. 9: The CDF of instantaneous queuing delay during heavy congestion

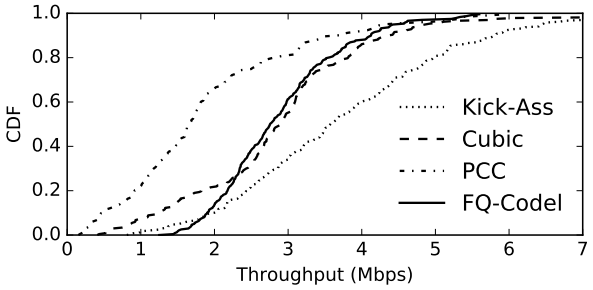

Fig. 10: The average throughput achieved by each flow over the duration during heavy congestion.

is able to complete flows on average $2\times$ faster, as Kick-Ass is able to readily obtain feedback from the router and fill the bandwidth accordingly. PCC, on the other hand, suffers longer flow completion times. The figure also shows that FQ-CoDel does not manage to improve performance in these scenarios as it is incapable of sending explicit rate feedback to endpoints.

We repeat the mixed TCP and Kick-Ass flow cases from Section III-D1 in the setting of our test-bed, again finding that Kick-Ass does not interfere with the completion times of TCP flows. We omit the results here due to space considerations.

Heavy Load Here, we compare the performance of Kick-Ass to other endpoint-based algorithms in cases of heavy load. In this setting, we further reduce the outgoing router link to 10Mbps and increase the average flow size to 1 MB and inter-flow times to a mean of 8 seconds.

Figure 9 shows a CDF of the instantaneous queuing delay of Kick-Ass compared to PCC, CUBIC (with a drop-tail queue), and CUBIC on FQ-Codel with ECN. We see that Kick-Ass is able to maintain extremely small queues for the duration, experiencing only small queueing delay, in line with what was seen in simulation, and avoiding the above 1 second queuing delays seen by CUBIC and PCC.

Figure 10 shows a CDF of the average throughput of flows. It demonstrates that Kick-Ass was able to take advantage of available throughput in the network, outperforming CUBIC. This is the case, independently of whether Drop-Tail or FQ-CoDel with ECN is used at the router. Moreover, Kick-Ass further significantly outperforms PCC.

B. Wide-Area Internet Performance

Here, we deploy Kick-Ass receivers at 10 remote locations on the Internet. A subset are deployed within the same city as the senders, while others are deployed at more distant locations, creating both intra- and inter-continental network paths. Our goal is to see if it is possible for Kick-Ass to achieve performance benefits on real wide-area Internet paths. For each destination, we generate Kick-Ass flows towards a receiver for 2 minutes; then we run TCP flows towards the same receiver for the next 2 minutes. By repeating this interleaving procedure over relatively short time scales, we aim to avoid any longer time-scale bias (*e.g.*, heavy network congestion while experimenting with one protocol. In all cases, we generate flows of the size between 1 and 2000 packets and measure flow completion times. On the sender side, we limit the Kick-Ass router's link capacity to 10 Mbps and 100 Mbps.

Kick-Ass outperforms TCP between 1.2 and $4\times$, depending on the nature of the given paths and the access link bandwidth. For a 10 Mbps Kick-Ass link, Kick-Ass consistently outperforms TCP in all scenarios. The Kick-Ass performance benefits are below one order of magnitude achievable in higher bandwidth scenarios (*e.g.*, 100 Mbps). Even in such cases, Kick-Ass does not aggressively queue as TCP does (*e.g.*, as observed in the case of 4G LTE wireless networks [17]), and hence achieves much more stable delay performance. For a 100 Mbps access link, Kick-Ass consistently outperforms TCP, yet the variance of the benefits is necessarily larger. When the available bandwidth on the path is larger, Kick-Ass flow completion times are expectedly much shorter, and hence the difference between Kick-Ass and TCP flows is more pronounced. When destinations are deeper in the Internet, it is more often the case that the bottleneck is not the edge Kick-Ass router's link, but rather a non-Kick-Ass link further away from the sender. In such cases Kick-Ass moves to TCP mode.

To validate this, we measure the percent of time that longer-lived flows stay in the Kick-Ass mode, *i.e.*, does not switch to the failover TCP mode. We evaluate a path where the destination is on the same continent as the sender, and the geographic distance between the two is approximately 2,000 miles. When the access link is set to 10 Mbps, it is obvious that the Kick-Ass link is the bottleneck because the flow stays entirely in the Kick-Ass mode. When the Kick-Ass access link is raised to 100 Mbps, the fraction of time spent in the failover TCP mode increases to 50% in the median case.

C. Packet Fragmentation Measurements

Here, we evaluate the state of the art regarding IP packet fragmentation on the Internet. Previous work from nearly 3 decades ago has suggested that fragmenting packets may be harmful [18], but many of the issues related to efficiency are not relevant to Kick-Ass, while other issues are no longer a concern with modern computing power. Other studies have explored how IP fragmentation behaves in the real Internet [19], [20]. While some of this work has had a different focus, they have encountered situations which seem to suggest that various network configurations and middle-boxes on a path may interfere with fragmented packets.

We perform an experiment to quantify the fraction of fragmented packets that encounter problems in the Internet. We

develop a simple web server which replies to client requests with packets of various sizes. It first sends increasing packet sizes, from 44 to 1480 bytes, sending each size 100 times, waiting for an acknowledgement after each packet is sent. After reaching the maximum size, the server begins sending 1480 byte packets which have been fragmented. We consider increasing sizes of the leading fragment, starting with 44 bytes (with the second fragment then being 1436, the remainder, and so on). We send each fragment size 100 times. Packets which are acknowledged by the client are considered successful, while those which time-out are assumed dropped.

To obtain a variety of clients from a large variety of locations, we conducted a large-scale experiment with Dasu [21]. Over the course of the experiment, 405 clients connected from 62 countries and completed the experiment, providing us with a broad view of the Internet edge. The server was operated on a machine on the local campus network.

We found that about 8.3% of clients failed to accept fragmented packets, often ending the connection as soon as it began sending fragmented packets. Given the diversity of our clients, and the fact that many are likely behind middle-boxes, firewalls, and home routers, this figure is not surprising. To cope with the possibility that traffic passing through a Kick-Ass router may be destined for such locations, Kick-Ass enabled routers respect Do-Not-Fragment flags in IP headers, routing such packets in the traditional manner. If a pair of endpoints are Kick-Ass enabled, they will refrain from setting the Do-Not-Fragment flag, allowing Kick-Ass routers to perform the necessary operations.

Drop Rates Given that a fragmented IP packet becomes two separate packets in the network, one expects that the probability that the packet will be dropped will approximately double, as only one of the fragments need drop for the entire packet to be lost. We find that the drop rate never exceeded twice the value of the full packet rate. These findings suggest that IP fragmentation provides a viable means of communication on the wider Internet.

RTT Encoding Kick-Ass utilizes packets in the range 600 to 611 bytes to communicate RTT from endpoints to routers, as explained in Section II-B above. In a trace loading the top 500 webpages, as well as a wireless capture in a busy cafe, we observed that less than .11% of TCP packets landed in this range. The use of a moving average and the extreme infrequency of natural packets of these sizes mean that this range of packets is safe for use as indicators.

Overhead The use of fragmentation as a communication mechanism is not without overhead. Each fragmentation introduces a new IP header, resulting in an additional 20 bytes of data. This reduces the total achievable utilization by 1.3% (*i.e.*, $20/1500$) in scenarios when all packets are fragmented. However, on the Internet, most bottlenecks occur on the network edge. Therefore, packets are likely to have a low rate set on the edge, allowing each packet to only be fragmented a single time, limiting the effect of these headers.

Fragmenting also introduces processing overhead, at routers and endpoints. However, this can be reduced via the mechanisms that limit the total number of packets that must be fragmented. As end-hosts and routers continue to become more powerful and have greater resources available, a greater amount of computing power can be devoted to congestion control and network management, as was seen in [1].

We used our testbed from Section IV-A to explore the time cost of of fragmentation to the end-hosts. In particular, we attempt to measure the time cost associated with packet fragmentation and reassembly. We consider two conditions: no fragmentation, and a fixed fragmentation on every packet. We then compare the difference in flow completion times for flows of increasing size, from 10 KB to 10 MB, repeating each condition on each size 30 times. We found no measurable difference in the times, suggesting that the ultimate performance bottle-neck lies elsewhere in the packet path.

V. Discussion

Overhead Reduction Header overhead can be further reduced by restricting the frequency of fragmentation at the router. In particular, a flow need not receive information at every single acknowledgement. In fact, the majority of these packets likely contain the same rate and are therefore redundant. The router could therefore restrict the number of packets it fragments by only fragmenting for particular time interval. Alternatively, the router could generate hashes for flows and attempt to only fragment packets once per RTT per flow, ensuring that it at worst overestimates the number of fragments needed. Such schemes would stand to further minimize overhead at the router [22].

IPv6 One component of the IPv6 specification is that it provides a mechanism for enabling end-hosts to perform end-to-end fragmentation, by way of a fragmentation extension header [23]. This provides IPv6 headers with the necessary fields to enable fragmentation (a fragment offset, more-fragments indicator, id number). This mechanism could be reused by routers, despite its usual prohibition in IPv6, without any side-effects for downstream routers or receivers. An IPv6-enabled Kick-Ass router would make use of these, providing the split packets with the appropriate header extensions, and performing these splits as in IPv4.

VI. Related Work

We provide a necessarily non-comprehensive overview of related work, pointing out a subset of prior work relating to Kick-Ass. Endpoint congestion control mechanisms have ranged from classic back-off mechanisms [24], [14], [25], [26], to complex analytic methodologies [27], [28], to RTT and delay based mechanisms [12], [29], [30]. More recently, work has been done to further improve the effectiveness of end point schemes, using machine learning to generate congestion control rules [1]. While often very effective, these approaches are limited by their lack of feedback from routers. Kick-Ass enables such router communication, while continuing to operate with these existing mechanisms.

Others have looked to routers to ease the problem of congestion, including the development of queue management

techniques [2], [31], [16]. Other systems have recognized the value in using information at the router to inform endpoints directly of congestion, [6], [3], [4], [5], [32], but have been limited by the necessity of compliance with traditional TCP/IP formats. Kick-Ass further builds off these works. Rather than providing an alternative algorithm for congestion control, it stands to enable the implementation of existing explicit rate schemes in the current Internet and the development of novel congestion control schemes that can be deployed within the Kick-Ass framework. In doing so, however, it does not prohibit the use of existing techniques, allowing the continued operation of time-proven congestion management schemes.

Our work further relates to cases where network mechanisms are used for purposes beyond their original design [33], [34]. For example, the use of port numbers in NAT firewalls, which effectively enlarged the IPv4 space [35]. We use IP fragmentation for congestion control. However, unlike the use of port numbers for NAT, our use of IP packet fragmentation is in agreement with internet design principles.

VII. Conclusions

In this paper, we presented Kick-Ass, a congestion control mechanism that enables the deployment of advanced explicit-rate congestion control protocols *within* the TCP/IP stack. We showed that packet sizes can be effectively utilized to implicitly communicate explicit-rate and other information from routers to endpoints and vice versa. Our large-scale testbed and Internet experiments demonstrated that Kick-Ass achieves superior performance in full-deployment scenarios, outperforming CUBIC on FQ-CoDeL and PCC. Furthermore, we showed that it retains high performance in partial-deployment scenarios, yet without causing any side-effects for the legacy traffic. Kick-Ass is incrementally deployable on the Internet. Endpoints and routers in operational networks can be seamlessly upgraded to support Kick-Ass. It thus enables a truly realistic pathway towards a faster Internet today. Beyond congestion control, we showed that embedding information into traffic patterns can be a powerful approach for overriding rigidity of internet protocols.

References

[1] K. Winstein and H. Balakrishnan, "TCP ex machina: Computer-generated congestion control," in *SIGCOMM '13*, 2013, pp. 123–134.

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

[3] S. Floyd, "TCP and explicit congestion notification," *SIGCOMM CCR '94*, vol. 24, no. 5, pp. 8–23, Oct. 1994.

[4] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. of ACM SIGCOMM '02*, 2002.

[5] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1281–1294, Dec. 2008.

[6] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the Internet," in *Proc. of IWQoS 2005*, 2005, pp. 271–285.

[7] NS-3, "Network simulator 3," http://www.nsnam.org.

[8] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *Proc. of USENIX NSDI'15*, April 2015.

[9] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *Proc. of ACM IMC '11*, 2011, pp. 181–194.

[10] R. Sinha, C. Papadopoulos, and J. Heidemann, "Internet packet size distributions: Some observations," University of Southern California Information Science Institute, Tech. Rep., May 2007. [Online]. Available: http://www.isi.edu/~johnh/PAPERS/Sinha07a.html

[11] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *SIGCOMM CCR '06*, vol. 36, no. 1, pp. 59–62, Jan. 2006.

[12] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan, "PCP: Efficient endpoint congestion control," in *Proc. of USENIX NSDI'06*, May 2006, pp. 197–210.

[13] D. Lacamera and H. Daniel, "MultiTCP," http://sourceforge.net/projects/multitcp/.

[14] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *SIGOPS '08*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[15] P. McKenney, "Stochastic fairness queueing," in *Proc. of IEEE INFOCOM '90*, 1990, pp. 733–740 vol.2.

[16] K. Nichols and V. Jacobson, "Controlling queue delay," *Queue*, vol. 10, no. 5, pp. 20:20–20:34, May 2012.

[17] J. Huang, F. Qian, Y. Guo, Y. Zhou, Q. Xu, Z. M. Mao, S. Sen, and O. Spatscheck, "An in-depth study of LTE: Effect of network protocol and application behavior on performance," in *Proc. of ACM SIGCOMM '13*, 2013, pp. 363–374.

[18] C. A. Kent and J. C. Mogul, "Fragmentation considered harmful," in *Proc. of ACM SIGCOMM '87*, 1988, pp. 390–401.

[19] E. Aben, "RIPE Atlas - packet size matters," https://labs.ripe.net/Members/emileaben/ripe-atlas-packet-size-matters, accessed: 2014-01-20.

[20] M. D. Boer, J. Bosma, B. Overeinder, and W. Toorop, "Discovering path MTU black holes on the internet using RIPE Atlas," Master's thesis, University of Amsterdam, 2012.

[21] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger, "Dasu: Pushing experiments to the internet's edge," in *Proc. of USENIX NSDI '13*, 2013, pp. 487–500.

[22] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based IP traceback," in *Proc. of ACM SIGCOMM '01*, 2001, pp. 3–14.

[23] S. Deering and R. Hinden, *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*, Internet Engineering Task Force, December 1998. [Online]. Available: http://tools.ietf.org/html/rfc2460

[24] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, Jun. 1989.

[25] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," in *Proc. of ACM SIGCOMM '96*, 1996, pp. 270–280.

[26] V. Jacobson, "Congestion avoidance and control," in *Proc. of ACM SIGCOMM '88*, 1988, pp. 273–288.

[27] D. Bansal and H. Balakrishnan, "Binomial congestion control algorithms," in *Proc. of IEEE INFOCOM '01.*, vol. 2, 2001, pp. 631–640.

[28] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. of ACM SIGCOMM '00*, 2000, pp. 43–56.

[29] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. of ACM SIGCOMM '94*, 1994, pp. 24–35.

[30] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. of MobiCom '01*, 2001, pp. 287–297.

[31] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management," in *Proc. of ACM SIGCOMM '01*, 2001, pp. 123–134.

[32] C.-H. Tai, J. Zhu, and N. Dukkipati, "Making large scale deployment of rcp practical for real networks," in *Proc. of IEEE INFOCOM '08*, 2008.

[33] V. Liu, A. Parks, V. Talla, S. Gollakota, D. Wetherall, and J. R. Smith, "Ambient backscatter: Wireless communication out of thin air," in *Proc. of ACM SIGCOMM '13*, 2013, pp. 39–50.

[34] J. Xiong and K. Jamieson, "Arraytrack: A fine-grained indoor location system," in *Proc. of USENIX NSDI'13*, 2013, pp. 71–84.

[35] P. F. Tsuchiya and T. Eng, "Extending the IP Internet through address reuse," *SIGCOMM CCR '93*, vol. 23, no. 1, pp. 16–33, Jan. 1993.